universidad del país vasco

euskal herriko unibertsitatea

Facultad de Informática / Informatika Fakultatea

API Tráfico en Tiempo Real

Alumno:	ENDIKA SÁNCHEZ GUTIÉRREZ
Tutor:	JESÚS BERMÚDEZ

Proyecto Fin de Carrera, Marzo 2014

ÍNDICE DE CONTENIDOS

ÍNDICE DE DIAGRAMAS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. Introducción

1.1 Descripción del proyecto

Viendo el creciente número de vehículos en nuestras carreteras, el número de retenciones, obras, accidentes…, la idea es crear un servicio que devuelva información sobre el estado del tráfico de las carreteras, de una forma fácil, para poder visualizarla en diferentes aplicaciones, y de esa forma poder gestionar mejor los viajes o los desplazamientos por carretera. Por ejemplo, en una empresa de transporte urgente, una furgoneta tiene que llevar un paquete a un destino en plena operación salida de Agosto, y tiene que pasar por Madrid. En esas fechas Madrid está muy colapsado de tráfico, y suele tener muchas retenciones. Por lo que se conectan a esta aplicación, comprueban el estado del tráfico, y deciden que ruta es la más optima, ahorrando así, tiempo y dinero, a un solo clic de ratón.

El desarrollo de este proyecto tiene como objetivo integrar en la plataforma Geoservicios[1] información relacionada con el tráfico, obtenida del distribuidor TomTom.

Para ello se creará un API en Java sobre dicha plataforma, en la cual se utilizarán y almacenarán los datos de tráfico obtenidos, para la visualización de la información en cualquier tipo de mapa. Junto con la visualización en tiempo real, también se desea guardar un histórico de toda esa información para su posterior análisis.

Este servicio está pensado para poder integrarse en diferentes tipos de aplicaciones. La demostración se hará sobre un visor web de mapas, pero también se puede integrar en aplicaciones de escritorio o en aplicaciones para smartphones.

1.1.1 Empresa

Geograma es una empresa especializada en Servicios y Consultoría de Topografía, Cartografía y Sistemas de Información Geográfica. Se encarga de cubrir todo el ciclo de vida de la información, desde la toma de datos en campo hasta la publicación web, pasando por la gestión y tratamiento de datos cartográficos.

Es la primera empresa en España que ofrece servicios con tecnología Mobile Mapping, es decir, la gama de soluciones “llave en mano”, que permiten la gestión completa del territorio a través de imágenes de 360º o panoramas.

Los servicios que ofrece el departamento al que pertenezco, GIS (Sistemas de Información Geográfica), son:

- *Formación*: impartir cursos a medida a sus clientes, para cubrir sus necesidades.
- *Desarrollos a medida*: realizar desarrollos atendiendo a las necesidades de los clientes.
- *Implantación GIS Corporativos*: se encarga de la dirección y ejecución de complejos proyectos de implantación de GIS Corporativos: Geosistemas y/o GIS especializados.

[1] Una plataforma que ofrece servicios geográficos a través de internet y resuelve problemáticas relativas al posicionamiento de direcciones postales, generación de rutas, obtención de mapas etc…

- *Geomarketing*: se refiere al uso del componente espacial contenido en los datos corporativos y del mercado, para la toma de decisiones empresariales inteligentes y más efectivas.

- *Visor Geográfico*: basado en la experiencia en la publicación de información geográfica en internet, tiene desarrollado el producto **Framework GS**. Se trata de un producto dirigido a la publicación sencilla de visores geográficos orientados a resolver diversas problemáticas en cualquier organización.

- *Geolocalización, geocoding*: La geocodificación es el proceso que permite obtener coordenadas a partir de direcciones postales. Las coordenadas obtenidas posibilitan la ubicación de los elementos en un mapa, y por tanto se puede comenzar a analizar gráficamente dichos elementos en función de variables geográficas: Cercanía o lejanía, área de influencia, densidad, relación con otros elementos, rutas de acceso, etc.

Entre otras, dispone de una plataforma llamada Geoservicios, desarrollada íntegramente por la empresa Geograma, S.L., la cual ofrece servicios geográficos a través de internet y resuelve problemáticas relativas al posicionamiento de direcciones postales, generación de rutas, obtención de mapas, etc.

Desde el año 1999 Geograma es partner distributor de TomTom (antigua Tele Atlas).
TomTom es líder mundial en la generación de cartografía digital. Esta cartografía se implanta en las empresas que tienen la necesidad de incluir la componente geográfica en sus procesos de negocio (banca, tasación, seguros, transporte, etc.).

1.1.2 Objetivos

El objetivo principal es desarrollar un módulo que se unirá a todos los servicios de la plataforma Geoservicios, que facilite el acceso a datos de información de tráfico proporcionado por TomTom, a los clientes que lo soliciten. El API que se desarrollará, deberá cumplir los siguientes objetivos:

- Desarrollar el proyecto siguiendo las especificaciones de la empresa y de la facultad. Con respecto a la facultad, se desarrollará una memoria siguiendo las pautas estudiadas en clase, y con respecto a la empresa, el servicio se realizará a tres niveles, SOAP, Servlet y API Javascript, junto a la documentación de usuario y desarrollo de los tres niveles.

- Llevar a cabo el plan desarrollado en este documento.

- Gestiónar los datos:
  - Obtener información sobre el estado de las carreteras, como obras, retenciones y circulación lenta.
  - Almacenar dicha información para crear un histórico de incidencias.

- Visualizar la información obtenida, en un visor de mapas alojado en la web.

- Finalizar el PFC dentro de los plazos estipulados y de forma exitosa.

1.1.3 Antecedentes

El primer análisis realizado tenía como objetivo la decisión de las tecnologías a utilizar para la presentación de los datos. En primer lugar se decidió utilizar Java entre el Feed de TomTom[1] y la plataforma de Geoservicios, ya que los datos obtenidos se presentan en OpenLR[2] y porque es un lenguaje de programación libre.

Para cumplir los tres niveles que se querían ofrecer, se decidió utilizar un servicio web SOAP[3] sobre Java, un Servlet y un API Javascript.

En el primer caso no se tuvo ninguna duda a la hora de decidir, ya que todos los servicios de la plataforma Geoservicios siguen el protocolo estándar SOAP. Se utilizan este tipo de servicios por su **extensibilidad** (pueden ser extendidos con nueva funcionalidad de forma sencilla y sin afectar a la ya existente), por su **neutralidad** (puede ser utilizado sobre cualquier protocolo de transporte como HTTP, SMTP, TCP o JMS), y sobre todo por su **independencia** (ya que permite cualquier modelo de programación).

El siguiente nivel, el servicio Servlet, se decidió gracias a que ya teníamos previos conocimientos sobre su funcionamiento, y simplemente se quiere utilizar como una pasarela entre una aplicación web y el servicio SOAP.

Para el tercer y último nivel, se utiliza Javascript. Básicamente se eligió ya que es una tecnología muy potente para trabajar sobre los navegadores, gracias a la potencia de procesamiento de estos, y porque ya teníamos conocimientos previos sobre esta tecnología.

- **Feed TomTom[1]:** es un servicio licenciado ofrecido por la compañía TomTom, que devuelve un documento xml, en el que se definen las diferentes incidencias de tráfico. Este documento, que esta accesible en internet pasando validación de IP y contraseña, se actualiza cada tres minutos.

- **OpenLR[2]:** es el nuevo estándar abierto de TomTom, con un bajo consumo de ancho de banda y que incrementará el potencial de los servicios de navegación y localización en cualquier mapa digital. Se trata de una tecnología de geoposicionamiento tanto dinámico como estándar, abierto para la industria de la navegación, mapas e ITS.

Su misión se centrará en los sistemas de información del tráfico y guías de rutas dinámicas, con la situación del tráfico o la meteorología como variantes a tener en cuenta. El que se parta de un estándar abierto sin pagos de por medio para el fabricante que quiera integrarlo ya es, por sí sola, una buena noticia.

- **SOAP[3]**: (*Simple Object Access Protocol*) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Es uno de los protocolos utilizados en los servicios Web.

En la parte de almacenamiento de datos, se barajaron dos posibles sistemas gestores de bases de datos, SQLite y Postgresql. El primero de ellos se pensó ya que es el SGBD que utiliza por defecto la librería OpenLR, y el segundo porque es el SGBD que se utiliza en la plataforma Geoservicios. Para obtener la localización de una incidencia del feed, es necesario un procesamiento de un dato binario mediante OpenLR, que se conecta con el mapa almacenado en la base de datos, y realiza una considerable suma de transacciones. Tras numerosas pruebas de rendimiento, se llego a la conclusión de que SQLite era más ligero con un mapa pequeño, como

podría ser el de una única ciudad, y por lo tanto obtenía la localización más rápido que Postgresql, pero en nuestro caso se quería utilizar el mapa de toda España, y en ese caso Postgresql era bastante más rápido.

A continuación se muestra una tabla con los diferentes resultados logrados al obtener diferente número de localizaciones utilizando el mapa de España con los dos SGBD.

	SQLite	**PostreSQL**
454 localizaciones	3min 36seg	2min 15seg
348 localizaciones	2min 52seg	1min 38seg
237 localizaciones	2min 14seg	1min 5seg
42 localizaciones	28seg	15seg

1.2 Método de trabajo

1.2.1 Métodos tácticos

Para realizar el trabajo de forma ordenada, se dividirá el mismo en distintas iteraciones, siguiendo el Proceso Unificado de Desarrollo (PUD). Para ello se establecerá con el tutor de la universidad el criterio a seguir, definiendo las diferentes fases del proyecto y sus correspondientes fechas de entrega

La comunicación con el tutor será principalmente por email, pero en caso que sea necesario se realizarán reuniones presenciales en su despacho.

Con respecto a la accesibilidad, la parte práctica, es decir, el desarrollo, despliegue y las pruebas se realizan en la empresa, y la parte más teórica, como la redacción de la memoria se podrá realizar entre horas libres y la empresa.

1.2.2 Métodos operativos

El trabajo se realiza por una sola persona, con ayuda, en caso de que así se requiera, del tutor del proyecto, del tutor de la empresa y del servicio técnico de TomTom.

Para el buen desarrollo del proyecto, en cuanto al hardware necesario, se necesita un ordenador con conexión a internet. Para el despliegue del servicio, también se necesita el servidor donde se aloja la plataforma de Geoservicios. En estos momentos, el servidor está alojado en Amazon.

Refiriéndonos al software, sistema operativo Window XP, con licencia de Microsoft Office. Para el desarrollo, se utiliza el software de Oracle, Eclipse. El SGBD, es

Postgresql, y para un acceso más sencillo, se utiliza PGAdmin. Finalmente, la zona servidora tiene sistema operativo Linux, pero no tendremos problemas de compatibilidad, ya que el desarrollo se realizará en Java.

Este proyecto, necesita de previos conocimientos en lenguajes informáticos, como de administración de bases de datos, y en menor proporción, pero por ello no menos importante, conocimientos de GIS (Sistemas de información geográfica). Durante el transcurso del trabajo, estos conocimientos se amplían considerablemente, y se aprende a utilizar otro tipo de tecnologías.

Aparte de lo ya comentado, son necesarias otras consideraciones, como el acceso a la plataforma privada de Geoservicios, mediante usuario, contraseña y un certificado, o el acceso a la información ofrecida por TomTom, que pasa por una subscripción de Geograma, obteniendo así una clave y validación por IP, por lo que esa información solo será consultable desde la plataforma o la oficina de la empresa.

1.3 Alcance

El ámbito de aplicación de este proyecto es educacional, y a la vez, una solución fácil e interesante para empresas o particulares que puedan necesitar información sobre el estado de nuestras carreteras. Esta información puede resultar muy útil para el cálculo de rutas por carreteras menos congestionadas, que nos pueden hacer ahorrar tiempo y dinero, para la consulta de carreteras conflictivas, incluso para crear historiales para su posterior consulta.

1.4 Planificación

1.4.1 Diagrama ETD

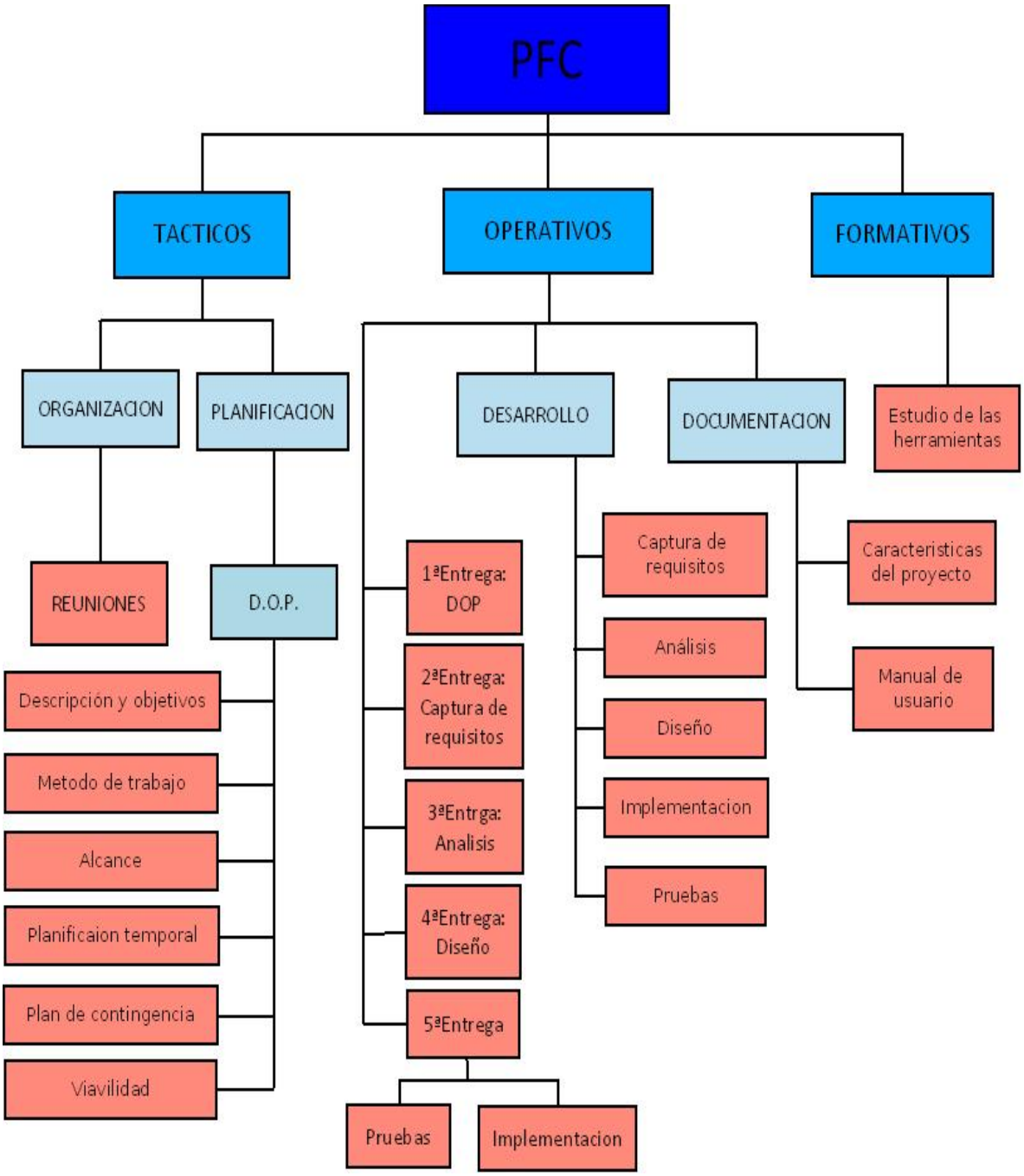


Diagrama 1: EDT

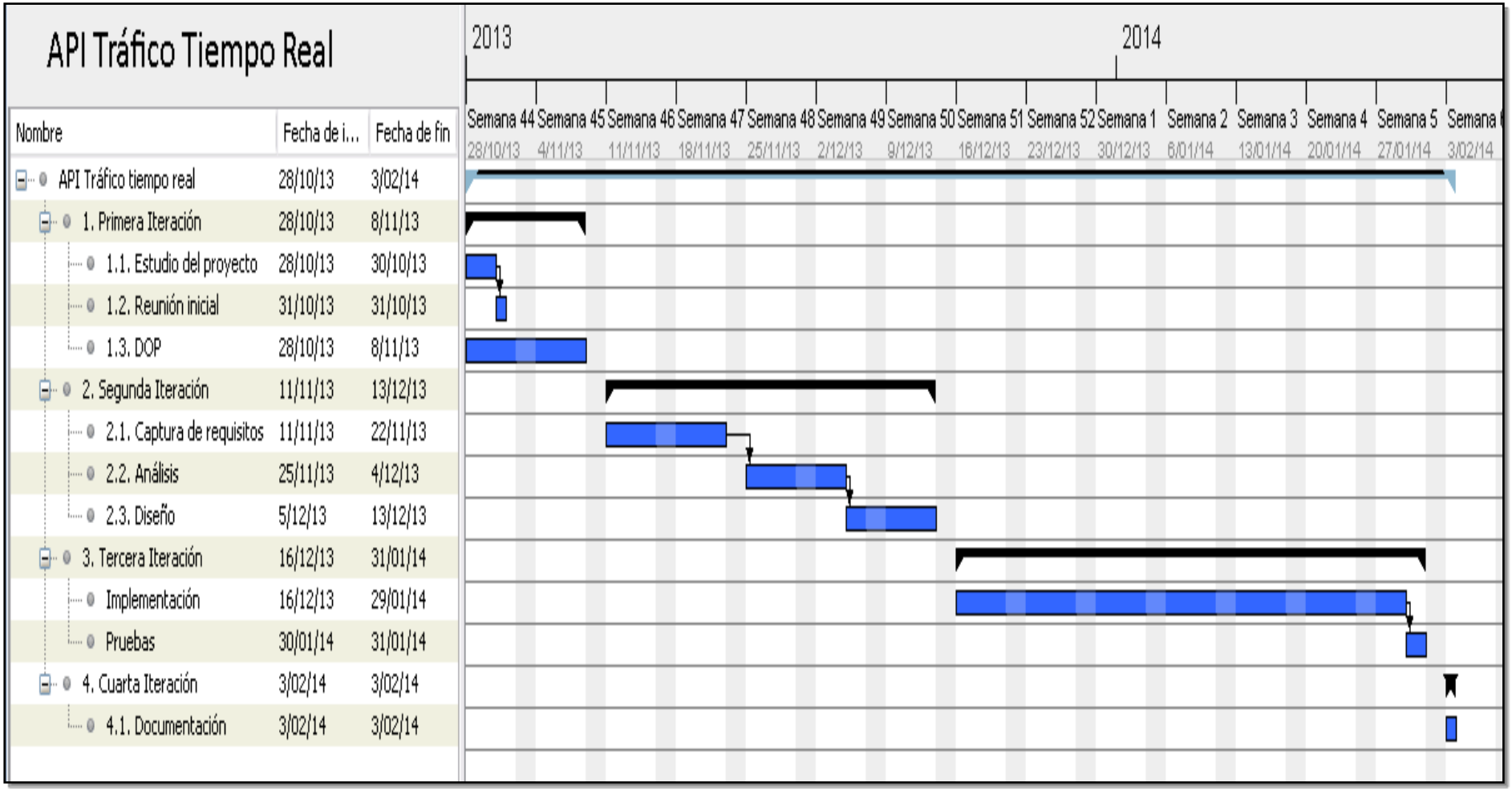
1.4.2 Planificación de entregas

La planificación de las fechas de entrega, y tiempos de desarrollo, no deja de ser una aproximación a los objetivos que se desean conseguir, por lo tanto es probable que existan adelantos o retrasos en alguna de las entregas detalladas a continuación:

	FASES	FECHA INICIO	FECHA FIN	DURACIÓN	TRABAJO
0	**0 - Estudio del proyecto y documentación**	**01-10-2013**	**Sin fecha de entrega**	**28 días**	**20 horas**
1	**1 – D.O.P.**	**28-10-2013**	**08-11-2014**	**8 días**	**16 horas**
1.1	1.1 – Descripción	28-10-2013	29-10-2013	2 día	4,5 horas
1.2	1.2 – Objetivos	30-10-2013	30-10-2013	1 día	2 horas
1.3	1.3 – Método de trabajo	31-10-2013	01-11-2013	1 día	1 hora
1.4	1.4 – Alcance	04-11-2013	04-10-2013	1 día	3 horas
1.5	1.5 – Planificación temporal(Diagrama de Gantt)	05-11-2013	06-10-2013	1 día	4 horas
1.6	1.6 – Plan de Contingencia	07-11-2013	07-10-2013	1 día	1 hora
1.7	1.7 – Factibilidad	08-11-2013	08-11-2013	1 día	0,5 hora
2	**2 – Captura de requisitos**	**11-11-2013**	**22-11-2013**	**10 días**	**28 horas**
3	**3 – Análisis**	**25-11-2013**	**04-12-2013**	**10 días**	**30 horas**
3.1	3.1 – Requerimientos	25-11-2013	26-11-2013	2 días	5 horas
3.2	3.2 – Modelos de casos de uso (MCU)	27-11-2013	30-11-2013	4 días	10 horas
3.3	3.3 – Modelo de dominio	01-12-2013	04-12-2013	4 días	15 horas
4	**4 – Diseño**	**05-12-2013**	**13-12-2013**	**7 días**	**16 horas**
4.1	4.1 – Diagramas de interacción	05-12-2013	08-12-2013	3 días	6 horas
4.2	4.2 – Diagrama de clases	09-12-2013	11-12-2013	2 días	4 horas
4.3	4.3 – Casos de uso reales	11-12-2013	13-12-2013	2 días	6 horas
5	**5 – Desarrollo y pruebas**	**16-12-2013**	**31-01-2014**	**47 días**	**150 horas**
5.1	5.1 - Desarrollo	16-12-2013	29-01-2014	44 días	144 horas
5.3	5.2 - Pruebas	30-07-2014	03-02-2014	3 días	6 horas
TOTAL		*01-10-2013*	*03-02-2014*	*110 días*	*248 horas*

Tabla 1: Planificación entregas

1.4.3 Diagrama de Gantt



Nombre	Fecha de i...	Fecha de fin
API Tráfico tiempo real	28/10/13	3/02/14
1. Primera Iteración	28/10/13	8/11/13
1.1. Estudio del proyecto	28/10/13	30/10/13
1.2. Reunión inicial	31/10/13	31/10/13
1.3. DOP	28/10/13	8/11/13
2. Segunda Iteración	11/11/13	13/12/13
2.1. Captura de requisitos	11/11/13	22/11/13
2.2. Análisis	25/11/13	4/12/13
2.3. Diseño	5/12/13	13/12/13
3. Tercera Iteración	16/12/13	31/01/14
Implementación	16/12/13	29/01/14
Pruebas	30/01/14	31/01/14
4. Cuarta Iteración	3/02/14	3/02/14
4.1. Documentación	3/02/14	3/02/14

Diagrama 2: Diagrama de Gantt

1.5 Plan de contingencia

El siguiente plan de contingencia se aplicará para reducir o anular el impacto en el presente proyecto de los diferentes problemas o situaciones anómalas que puedan surgir durante la realización del mismo.

1.5.1 Análisis y evaluación de riesgos

1.5.1.1 Problemas del alumno

-Caso 1:

-***Escenario***: Ausencia del alumno para la elaboración del proyecto, por motivos ajenos al proyecto.

-***Impacto***: **Medio.** La tarea asignada a dicha fecha no podrá ser realizada.

-***Estrategia a seguir***: Sacar horas extra para seguir la planificación establecida. En caso extremo, avisar al tutor del proyecto.

-***Procedimiento***: Sacar tiempo para la dedicación al PFC.

-***Probabilidad***: Pequeña.

-***Gravedad***: 1 día.

-Caso 2:

-***Escenario***: Ausencia prolongada del alumno, por enfermedad.

-***Impacto***: Alto. Las tareas asignadas a dichas fechas no podrán ser realizadas.

-***Estrategia a seguir***: Avisar al tutor de la incidencia.

-***Procedimiento***: Se replantearían los plazos de entrega acordados inicialmente.

-***Probabilidad***: Pequeña.

-***Gravedad***: 7 días o más.

-Caso 3:

-***Escenario***: Demora en el desempeño de alguna tarea.

-***Impacto***: Alto. La tarea correrá el peligro de ser entregada tarde.

-***Estrategia a seguir***: Reajustar el horario establecido con el fin de sacar mayor rendimiento diario.

-***Procedimiento***: Re planificar las horas estipuladas.

-***Probabilidad***: Media.

-***Gravedad***: 1-3 días.

1.5.1.2 Problemas de hardware

-Caso 1:

-***Escenario***: Ordenador personal averiado.

-***Impacto***: **Alto**. La tarea no se podría desempeñar en el ordenador, posibles pérdidas de información o de parte del proyecto.

-***Estrategia a seguir***: Tener acceso a otro ordenador, y hacer copias de seguridad de los datos periódicamente.

-***Procedimiento***: Recuperar el ultimo backup, copiarlo al nuevo ordenador y continuar la realización del proyecto.

-***Probabilidad***: Pequeña

-***Gravedad***: 1-4 días

1.5.1.3 Problemas de software

-Caso 1:

-***Escenario***: Pérdida de datos parcial o total debida a un virus, error del SO.

-***Impacto***: **Alto**. Dicha parte correrá el peligro de ser perdida.

-***Estrategia a seguir***: Se intentará recuperar los datos con las diferentes herramientas que sean necesarias.

-***Procedimiento***: Buscar en las copias de respaldo dicha información, de no estar, tendrá que asumirse la pérdida y la necesidad de volver a realizar esa parte.

-***Probabilidad***: Pequeña.

-***Gravedad***: 1- 7 días (7diás fecha de realización de copias de seguridad).

1.5.1.4 Problemas de implementación

-Caso 1:

-***Escenario***: No se puede continuar por falta de conocimientos.

-***Impacto***: **Medio.**

-***Estrategia a seguir***: Intentar documentarse con anterioridad.

-***Procedimiento***: En caso de no conseguir resolver el problema, pedir ayuda al tutor o alguna persona experta en el tema, y de no ser posible, buscar otra alternativa a la problemática.

-***Probabilidad***: Media.

-***Gravedad***: 1-2 días.

-Caso 2:

-***Escenario***: Al reunir las partes, no funciona correctamente.

-***Impacto***: **Medio**. Se detendrán las pruebas de las partes afectadas.

-***Estrategia a seguir***: Hacer un buen diseño para que esto no ocurra.

-***Procedimiento***: Buscar los errores de implementación que generen los errores.

-***Probabilidad***: Alta.

-***Gravedad***: 1-5 días.

1.5.2 Riesgos críticos

Analizaremos soluciones viables cuando los problemas planteados sean de una gravedad importante, y las soluciones no sean triviales:

-Se tendrá que organizar una reunión urgente, lo antes posible.

-Se abordará el problema para comprender su magnitud.

-Se plantearán todas las soluciones para seleccionar la más óptima.

1.6 Factibilidad

Después de realizar el DOP y estudiar todos los puntos que se deben tener en cuenta para realizar el proyecto, hemos comprobado que disponemos de los recursos humanos y materiales necesarios para llevarlo a cabo, por lo cual consideramos factible la materialización de dicho proyecto y el cumplimiento de los plazos de entrega acordados.

2. Captura de requisitos

La documentación que viene a continuación presenta el Modelo de Casos de Uso (MCU) y el Modelo de Dominio (MD) para la gestión de datos de tráfico en tiempo real, tanto para la parte servidora encargada de procesar esa información, como para la parte cliente que se encarga de mostrar dicha información en una aplicación web. Además, se añaden los casos de uso con una descripción de alto nivel.

2.1 Descripción interfaz de usuario

La interfaz de usuario será de consulta y no interactuará directamente con el servicio creado. Para ello, sobre una aplicación web, constará de dos zonas principales. Una de ellas, será la zona del mapa, donde se le mostrarán las incidencias en el lugar que corresponda con un icono que diferenciará el tipo de incidencia. La otra zona, será un panel en la parte derecha de la pantalla, en que se mostrará un listado con todas las incidencias mostradas en el mapa, con su descripción asociada. Ambas partes, interactuarán entre ellas ya que cuando el usuario pase el ratón por encima de una incidencia en el mapa, ésta se seleccionará en el panel de la derecha, y por el contrario, cuando el usuario pase el puntero por encima de una incidencia en el panel, se seleccionará la incidencia del mapa.

2.2 Modelo de casos de uso

Un usuario anónimo accede a la aplicación web de tráfico en tiempo real, en ese momento, la aplicación se inicializa, y muestra una zona con un mapa y otra zona de paneles. Este proceso no lo tendremos en cuenta en este proyecto ya que para ello se utiliza una librería ya creada anteriormente, que se compone por un motor o núcleo y por unos componentes. Una vez inicializada, se carga sobre el mapa una capa de incidencias, para ello esta capa tendrá que hacer una petición a un servicio para obtener las incidencias y así poder mostrarlas sobre el mapa. Esta capa se actualizará cada tres minutos para mantenerse actualizada. A parte de la capa de incidencias, se carga también un panel con el listado de incidencias de tráfico visibles en ese momento sobre el mapa.

Una vez que este todo cargado, el usuario puede pasar el puntero del ratón por encima de una incidencia del mapa, en ese momento, se seleccionará esta incidencia en el listado que está en el panel, y se mostrará su información asociada en la parte inferior de dicho panel. A su vez, el usuario puede seleccionar una incidencia sobre el panel colocando el ratón encima, la incidencia sobre el mapa cambiará su estado de visibilidad, se creará un icono más grande, y se mostrará la información asociada en la parte inferior del panel. Finalmente, el usuario podrá hacer doble clic sobre un elemento de la lista de incidencias y el mapa hará zoom sobre la incidencia asociada del mapa.

El tiempo también tendrá un papel importante sobre el visor web, ya que la capa de incidencias del mapa tiene que estar actualizada. Para ello, se programará un temporizador que hará que se actualice la información de la aplicación web cada tres minutos.

Para que esa información actualizada esté disponible, se configurará en el administrador regular de procesos en segundo plano (demonio) del servidor Linux,

llamado 'cron', una tarea que cada tres minutos también lance un proceso de actualización de los datos. De esa forma mantendremos en una base de datos la información sobre incidencias de tráfico lo más actualizada posible.

Después de describir el funcionamiento general del sistema de tráfico en tiempo real, podemos observar varios actores externos al sistema para el modelo de casos de uso. En primer lugar tenemos al usuario final, quien interactuará con la aplicación web, y en segundo lugar podríamos deducir que el tiempo, aun no siendo un ente humano, sería nuestro segundo actor externo, ya que de él depende que el sistema mantenga las incidencias de tráfico actualizadas en todo momento.

Figura 1: Actores

2.2.1 Casos de uso del Usuario

En el siguiente diagrama[1] definimos los casos de uso relativos al actor "Usuario". En primer lugar, el usuario accederá a la aplicación web (1.Acceso al visor), inicializando así una serie de procesos para solicitar y mostrar la información sobre el visor web. Una vez esté la información cargada en el navegador, el usuario podrá situar el cursor sobre una incidencia del mapa, esta se seleccionará y se mostrará su información asociada sobre el panel de la derecha (2.Ver información). Así mismo, el usuario podrá situar el cursor sobre una incidencia en el panel de la derecha y de esa forma se seleccionará la incidencia sobre el mapa y se mostrará su información asociada (3. Seleccionar incidencia). Para hacer zoom a una incidencia, el usuario podrá hacer doble clic en el listado de incidencias del panel de la derecha (4.Zoom a incidencia).

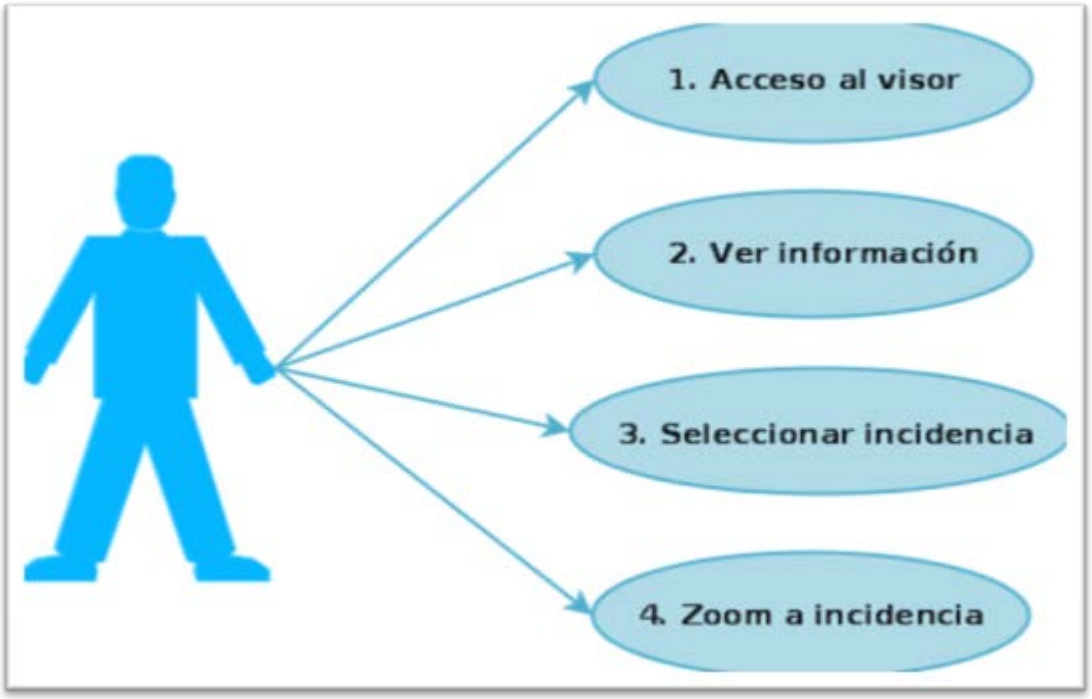


Diagrama 3: Casos de uso "Usuario"

2.2.1.1 Caso de uso "Acceso al visor"

En este primer caso de uso del usuario, el usuario simplemente accede al visor mediante un navegador web, lo que desencadena que se lancen una serie de procesos, entre ellos uno que corresponde al segundo actor definido, el tiempo. Para ello, se abre el navegador preferido e se introduce la URL. En este caso, no necesita validación.

Caso de uso expandido

Actores: *Usuario*, *Tiempo*

Pre: El *Usuario* dispone de internet en su equipo y conoce la URL de acceso.

Post: Se muestra en la pantalla el visor web, con un panel a la derecha y su listado de incidencias correspondientes, y con dichas incidencias representadas sobre un mapa dibujado en otro panel central. Se lanza un proceso que cada tres minutos actualizará dichas incidencias.

Resumen: El usuario accede a la aplicación web y se les muestran el panel del mapa y el panel con el listado de incidencias, y la aplicación inicia un proceso de actualización.

Escenario principal o curso normal de los acontecimientos

1. ***Usuario***: Abre el navegador e introduce la URL
2. **Aplicación Web**: Carga la aplicación, solicita las incidencias para dibujarlas sobre el mapa y crear el listado, y lanza un proceso que actualizará la información cada tres minutos.
3. ***Tiempo***: Pasados tres minutos, vuelve a solicitar incidencias "frescas", para que la aplicación web actualice sus datos. Este paso, se repite indefinidamente cada tres minutos hasta que se cierre la aplicación.

Extensiones o cursos alternativos

Paso 2: Ocurre algún error en la carga de la aplicación. Se muestra un mensaje de información sobre el error, si el error es grave la aplicación no se carga, si es leve, la aplicación se cargará.

Paso 3: Ocurre algún error al actualizar los datos. La aplicación se mantendrá sin datos durante tres minutos hasta la próxima actualización.

2.2.1.2 Caso de uso "Ver información"

Una vez hayamos accedido al visor web, el usuario desea ver la información asociada a una incidencia, para ello coloca el puntero sobre la incidencia en el mapa. La aplicación selecciona dicha incidencia cambiándole el estilo al icono, lo cambia por uno más grande, y muestra su información asociada en la zona inferior del panel de la derecha. Cuando el usuario retira el puntero de la incidencia, ésta se deselecciona y se limpia el contenido del panel de la derecha.

Caso de uso expandido

Actores: *Usuario*

Pre: La aplicación está cargada en el visor, y existen incidencias en ese momento.

Post: Se muestra la información asociada a la incidencia seleccionada en el panel de la derecha

Resumen: El *Usuario* coloca el cursor sobre una incidencia del mapa, ésta se selecciona y se muestra su información asociada en el panel de la derecha. Cuando el *Usuario* retira el cursor, se deselecciona y se limpia el panel.

Escenario principal o curso normal de los acontecimientos

1. ***Usuario***: Sitúa el puntero del ratón sobre una incidencia del mapa.
2. **Aplicación Web**: Selecciona la incidencia cambiándole el icono, y carga en el panel de la derecha la información asociada.
3. ***Usuario***: Retira el puntero de la incidencia.
4. **Aplicación Web**: Deselecciona la incidencia, le vuelve a cambiar el icono al más pequeño, y limpia el panel de información de la derecha.

Extensiones o cursos alternativos

Pasos 1, 2, 3, 4: Ocurre algún error inesperado en alguno de los pasos. La aplicación no seleccionará la incidencia y no mostrará su información asociada sobre el panel.

2.2.1.3 Caso de uso "Seleccionar incidencia"

Este caso de uso es muy similar al anterior, con la diferencia de que en este caso el usuario coloca el puntero del ratón sobre una incidencia del listado de incidencias. A continuación se selecciona la incidencia sobre el mapa cambiando el tamaño del icono asociado, y se muestra la información en la parte inferior del listado.

Caso de uso expandido

Actores: Usuario.

Pre: El usuario ha accedido al visor y existen incidencias.

Post: Se selecciona la incidencia sobre el mapa, y se muestra la información asociada en la parte inferior del panel de resultados.

Escenario principal o curso normal de los acontecimientos

1. ***Usuario***: Sitúa el puntero del ratón sobre una incidencia en el listado de incidencias del panel de la derecha.

2. **Aplicación Web**: Selecciona la incidencia sobre el mapa cambiándole el icono y muestra su información asociada en la zona inferior del panel.

3. ***Usuario***: Retira el puntero de la incidencia sobre el listado.

4. **Aplicación web**: Deselecciona la incidencia del mapa poniéndole el icono por defecto, y limpia la información del panel.

Extensiones o cursos alternativos

Pasos 1, 2, 3, 4: Ocurre algún error inesperado en alguno de los pasos. La aplicación no seleccionará la incidencia y no mostrará su información asociada sobre el panel.

2.2.1.4 Caso de uso "Zoom a incidencia"

El usuario desea ver el tramo al que afecta una incidencia de forma más clara. Para ello hace doble clic sobre una incidencia del listado de incidencias. El sistema busca la incidencia sobre la que el usuario ha hecho doble clic y aplica un zoom a esa zona, de esta forma en el mapa se visualizara solo la zona a la que afecta la incidencia.

Caso de uso expandido

Actores: *Usuario.*

Pre: El *usuario* desea ver el tramo afectado por una incidencia.

Post: El mapa se ajusta a la zona afectada por la incidencia.

Resumen: El *usuario* hace doble clic sobre una incidencia del listado, el sistema recibe el evento y se ajusta al zoom de la incidencia deseada.

Escenario principal o curso normal de los acontecimientos

1. ***Usuario***: Hace doble clic sobre una incidencia del listado de incidencias.

2. **Aplicación web**: Recibe el doble clic y ajusta la extensión del mapa a la incidencia seleccionada.

Extensiones o cursos alternativos

Paso 1: El *usuario* ha colocado el ratón sobre la incidencia, por lo que se lanza el proceso del caso de uso 3.

2.2.2 Casos de uso del "Tiempo"

El actor tiempo, tiene mucha relevancia en este proyecto, ya que es muy importante que toda la información se encuentre actualizada en todo momento. Tan importante es la parte asociada a la visualización de los datos, como la parte de almacenamiento de los datos. Como ya hemos definido anteriormente, cuando el usuario acceda al visor, se lanza un proceso temporizador que cada tres minutos refresca la información sobre el visor (1. Refrescar visor). A su vez, el proceso que

recupera los datos de TomTom y los almacena en la base de datos, también está configurado para que cada 3 minutos vuelva a actualizar la información de la base de datos (2. Actualizar datos). Por lo que nos encontramos con 2 casos de uso.

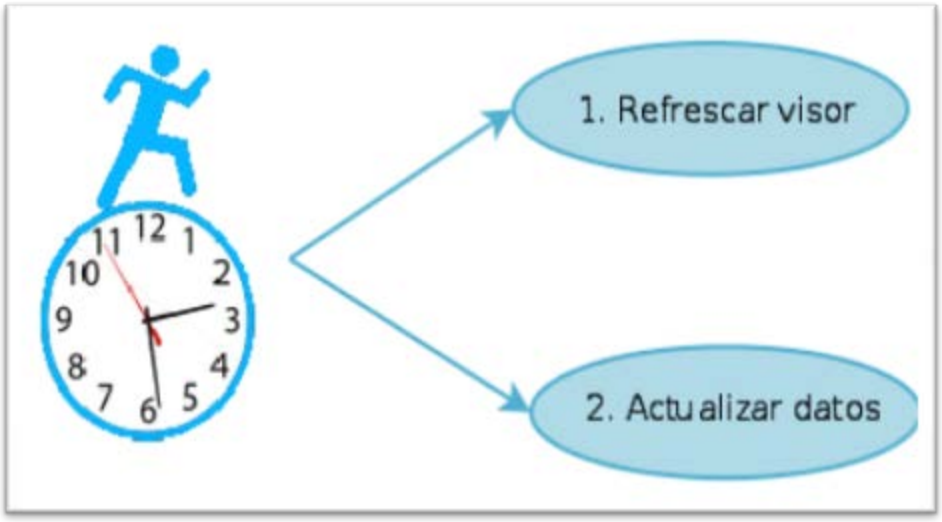


Diagrama 4: Casos de uso "Tiempo"

2.2.2.1 Caso de uso "Refrescar visor"

Cuando un usuario accede al visor, se le muestran las últimas incidencias de tráfico detectadas. Pasado un tiempo, si el usuario mantiene abierta la aplicación, esas incidencias deberían haberse actualizado. Para ello el actor "Tiempo" avisa al visor cada 3 minutos para que este realice el proceso de actualización de las incidencias de tráfico, de esta forma se mantiene el visor actualizado en todo momento.

Caso de uso expandido

Actores: *Usuario*, *Tiempo*

Pre: El usuario accede a la aplicación para consultar las incidencias de tráfico en ese momento

Post: El sistema mantiene actualizada la información de tráfico en todo momento.

Resumen: Desde que el usuario accede al visor, el actor tiempo avisa cada 3 minutos a la aplicación para que se mantenga actualizada

Escenario principal o curso normal de los acontecimientos

1. ***Usuario***: Accede a la aplicación web.
2. ***Tiempo***: Se inicia un temporizador.
3. ***Tiempo***: Pasados tres minutos, el temporizador avisa al visor para que se actualice, y vuelve a iniciar la "cuenta atrás".
4. **Aplicación Web**: Actualiza todas las incidencias de tráfico.

Extensiones o curso alternativos

Pasos 2, 3, 4: Ocurre algún tipo de error. El sistema no actualiza las incidencias, por lo que espera al próximo aviso del temporizador.

2.2.2.2 Caso de uso "Actualizar datos"

Para mantener los datos actualizados en la base de datos en todo momento, se define un temporizador en el servidor que cada tres minutos lanza un proceso. Este proceso borra los datos desactualizados de la base de datos, recupera las últimas incidencias de TomTom, obtiene el tramo afectado y guarda un registro por cada incidencia en la base de datos.

Caso de uso expandido

Actores: *Tiempo*

Pre: Esta desplegado un proceso en el servidor y se ha configurado un temporizador.

Post: Se actualizan los datos de la base de datos

Resumen: Cada tres minutos, el actor *Tiempo*, ejecuta un proceso que se encarga de actualizar todos los datos de la base de datos.

Escenario principal o cuso normal de los acontecimientos

1. ***Tiempo***: Cada tres minutos lanza un proceso y repite el paso dos de este caso de uso.
2. **Sistema**: El proceso se conecta con TomTom, lee las nuevas incidencias de tráfico y actualiza la base de datos

Extensiones o curso alternativo

Paso 2: Falla el proceso de actualización. Guarda en un fichero log la descripción del error, la base de datos se queda vacía y espera a la siguiente llamada. Si el fallo

ocurre solo con una incidencia, esta incidencia no se añade a la base de datos pero continúa.

2.2.3 Modelo de dominio

A continuación se expone el modelo de dominio del sistema completo de Tráfico en Tiempo Real. Más adelante se especificará más detalladamente cada parte del modelo, incluyendo submodelos específicos por cada apartado importante. Este modelo de dominio incluye, por una parte el acceso a los datos y por otra la visualización de dichos datos.

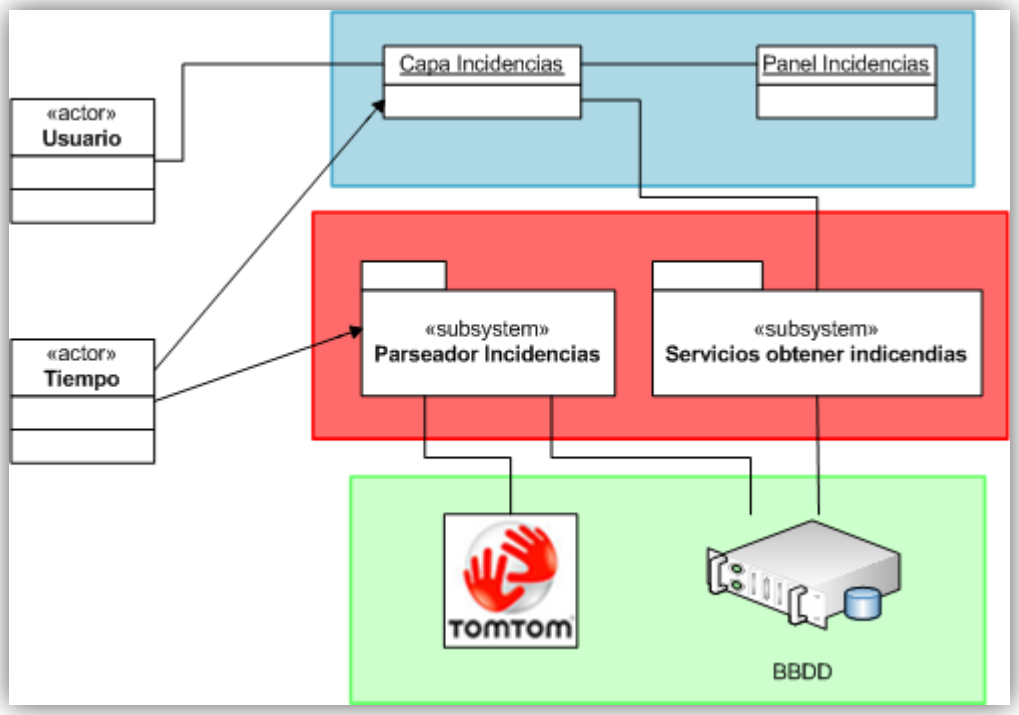


Diagrama 5: Modelo de dominio

El **usuario** puede acceder al visor, que automáticamente solicitará el listado de incidencias actuales mediante la **capa de incidencias**.

La **capa de incidencias**, se comunicará con el **panel de incidencias** para sincronizarse y mostrar las mismas incidencias. A su vez, **la capa de incidencias** se actualizará cuando el actor **tiempo** se lo indique, para ello se conectará con el subsistema de **servicios para obtener incidencias**.

El **panel de incidencias** estará constantemente sincronizado con la **capa de incidencias**, de esa forma en todo momento mantendrá un listado actualizado de incidencias.

Cada 3 minutos, el *actor* **tiempo** avisará a la **capa de incidencias** para que se actualice, y al mismo tiempo avisará al **parseador de incidencias** para que se actualice con los datos de **TomTom** y actualice la **BBDD**.

El subsistema **parseador de incidencias** estará a la espera de que el *actor* **tiempo** le indique que tiene que actualizar la **BBDD**, para ello primero se conectará con los datos de **TomTom** y a continuación actualizará la **BBDD**.

El subsistema de **obtención de incidencias**, estará a la espera de que la **capa de incidencias** le solicite la información, y en ese momento se tendrá que conectar con la **BBDD** para recuperar las incidencias actuales y devolverlas.

3. Análisis

A continuación vamos a exponer el análisis asociado a cada caso de uso presentado en el apartado anterior.

3.1 Casos de uso del "Usuario"

3.1.1 Caso de uso "Acceso al visor"

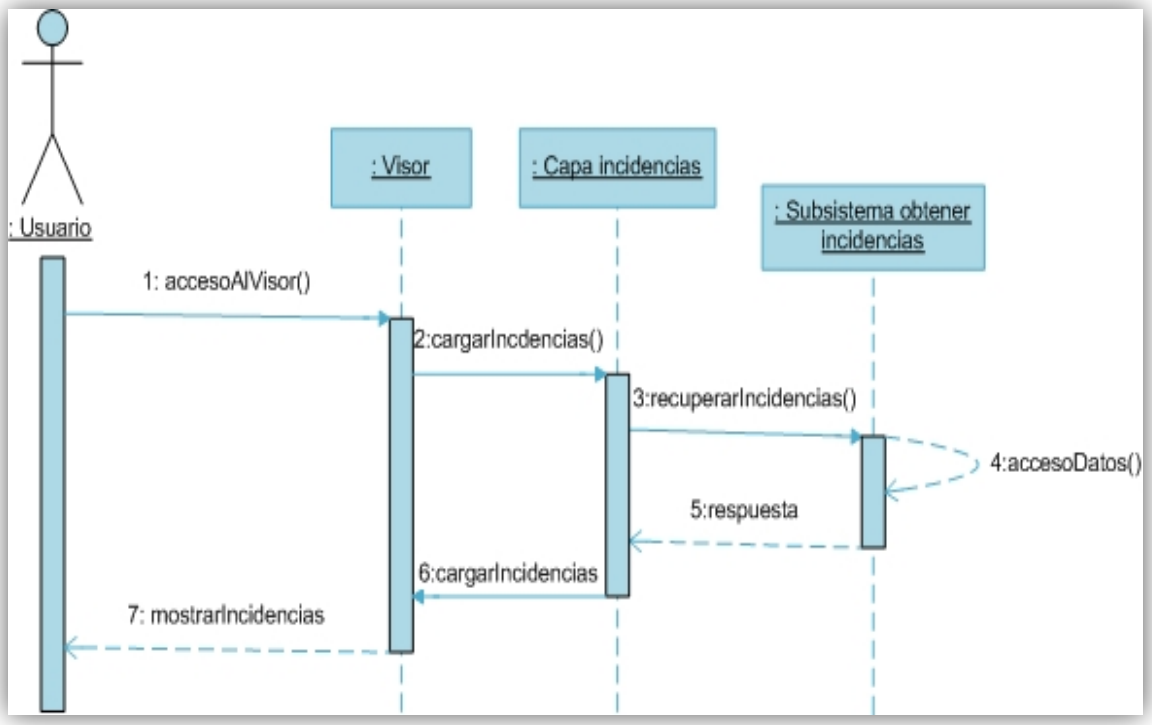


Diagrama 6: Caso de uso "Acceso al visor"

Operaciones

1. **Name**: accesoAlVisor
   Responsabilities: acceder al visor para visualizar las incidencias.
   Preconditions: Ø
   Postconditions: Ø
   Returns: Ø

2. **Name**: cargarIncidencias()
   Preconditions: debemos tener correctamente configurada la capa de incidencias.
   Postconditions: el visor debe mostrar la capa con las incidencias actuales.
   Returns: Ø

3. **Name**: recuperarIncidencias()
   Preconditions: la capa de incidencias está configurada con un usuario y contraseña válidos para la obtención de información.
   Postconditions: recibimos el listado de incidencias.
   Returns: listado de incidencias.

4. **Name**: accesoADatos()
   Preconditions: se tiene acceso a la tabla de la base de datos con las incidencias actualizadas.
   Postconditions: recibimos el listado de incidencias.
   Returns: listado de incidencias.

3.1.2 Caso de uso "Ver información"

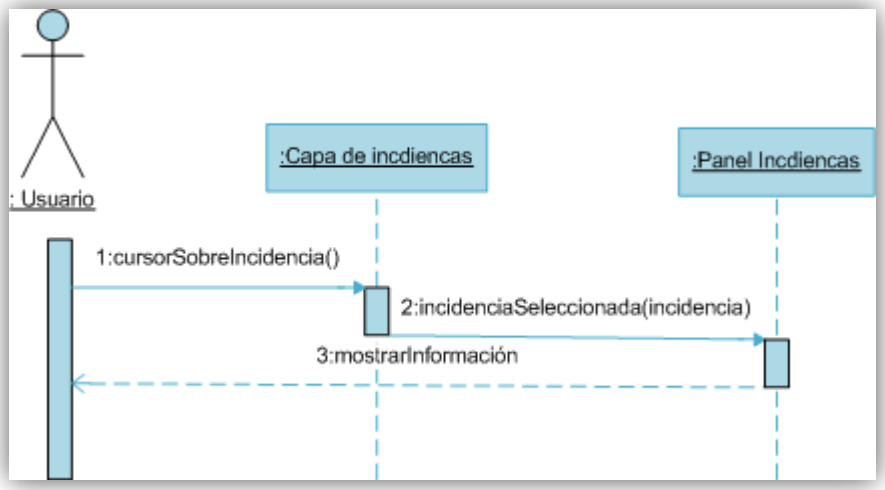


Diagrama 7: Caso de uso "Ver información"

Operaciones

1. **Name**: cursorSobreIncidencia()
   Preconditions: existe alguna incidencia cargada en el mapa.
   Postconditions: Ø
   Returns: Ø
2. **Name**: incidenciaSeleccionada()
   Preconditions: panel y capa de incidencias están sincronizados
   Postconditions: Ø
   Returns: Ø

3.1.3 Caso de uso "Seleccionar Incidencia"

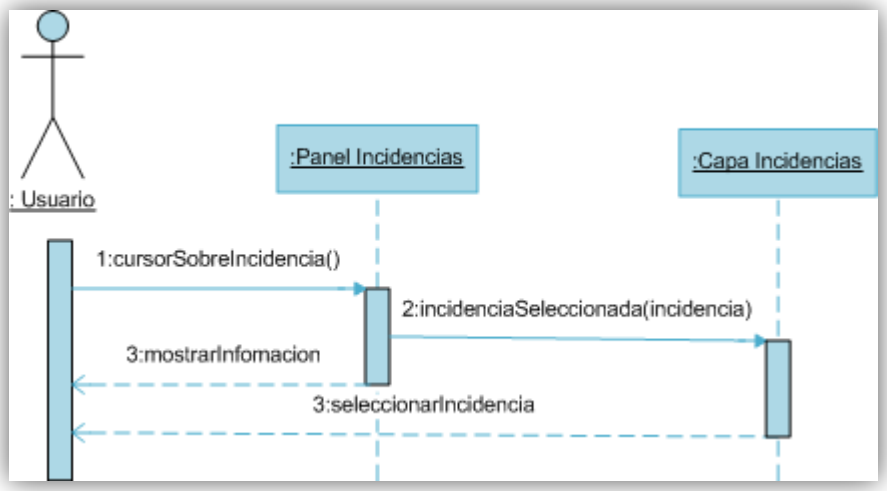


Diagrama 8: Caso de uso "Seleccionar incidencia"

Operaciones

1. **Name**: cursorSobreIncidencia()
   Preconditions: existe alguna incidencia cargada en el panel de incidencias.
   Postconditions: se muestra la información en panel inferior.
   Returns: información asociada a la incidencia.

2. **Name**: incidenciaSeleccionada()
   Preconditions: panel y capa de incidencias están sincronizados.
   Postconditions: Ø
   Returns: Ø

3.1.4 Caso de uso "Zoom a Incidencia"

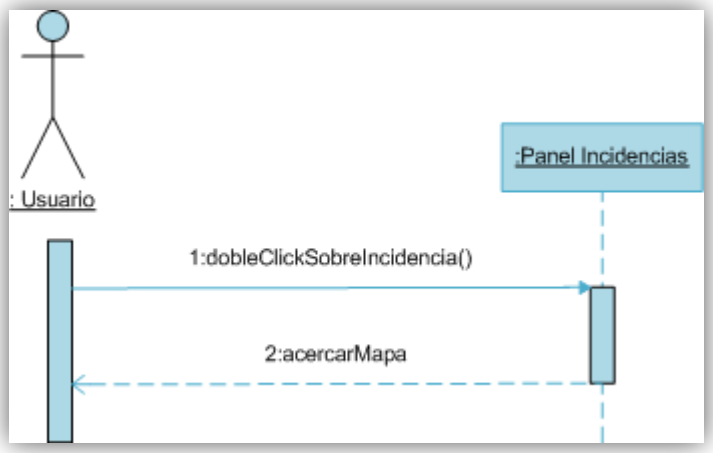


Diagrama 9: Caso de uso "Zoom a incidencia"

Operaciones

1. **Name**: dobleClickSobreIncidencia()
   Preconditions: existe alguna incidencia cargada en el panel de incidencias.
   Postconditions: se hace zoom a la extensión de la incidencia en el mapa.
   Returns: Ø

3.2 Casos de uso del "Tiempo"

3.2.1 Caso de uso "Refrescar Visor"

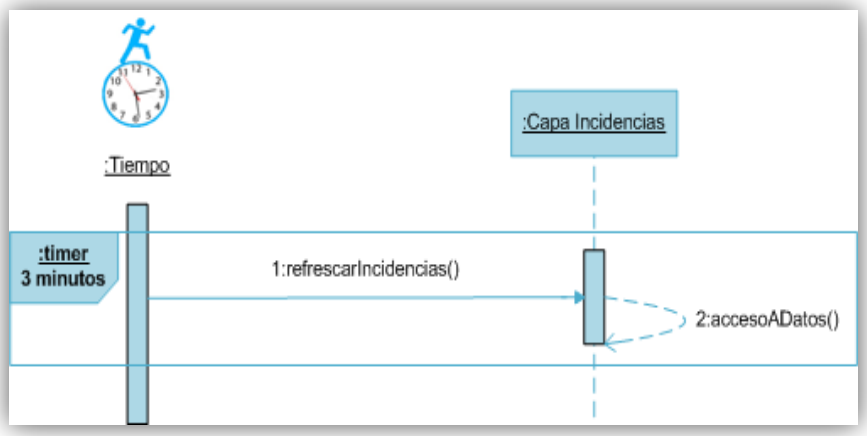


Diagrama 10: Caso de uso "Refrescar visor"

Operaciones

1. **Name**: refrescarIncidencias()
   Preconditions: la aplicación esta iniciada por un usuario.
   Postconditions: cada tres minutos se actualiza la capa de incidencias.
   Returns: Ø

3.2.2 Caso de uso "Actualizar Datos"

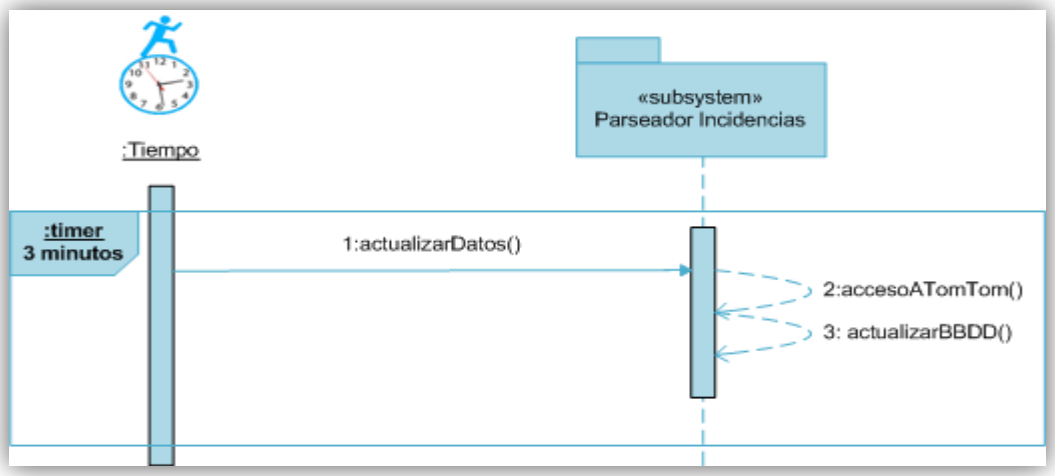


Diagrama 11: Caso de uso "Actualizar datos"

Operaciones

1. **Name**: actualizarDatos()
   Preconditions: el *subsistema parseador* tiene acceso a la BBDD y a la información proporcionada por TomTom.
   Postconditions: cada tres minutos se actualiza la tabla de incidencias.
   Returns: Ø

4. Arquitectura del Sistema

4.1 Arquitectura MVC

La arquitectura elegida para el sistema de "Tráfico en Tiempo Real" es la arquitectura denominada **modelo-vista-controlador (MVC)** adaptada a las necesidades propias del sistema.

Definición MVC:

*El **Modelo Vista Controlador (MVC)** es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.*

De manera genérica, los componentes de MVC se podrían definir como sigue:

- *El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.*

- *El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (**Middleware**).*

- *La **Vista**: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.*

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón, estos frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor.

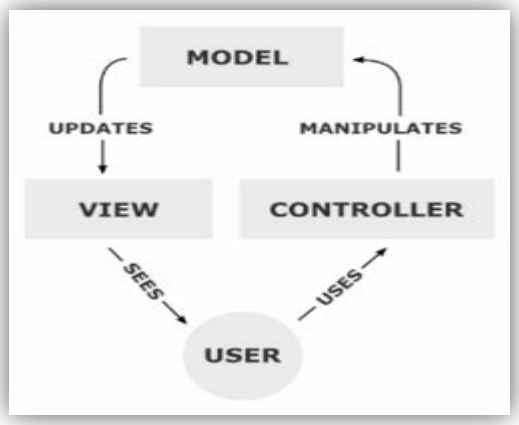


Figura 2: Modelo Vista Controlador

(Fuente: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador)

4.2 Elección de Tecnologías

Desde un principio se quería que el sistema estuviera formado por tres niveles independientes, el cliente, el servidor y los datos.

4.2.1 Capa de Presentación

Para el primer nivel, **cliente**, se utiliza un producto desarrollado por la empresa, llamado FrameworkGis y desarrollado con el lenguaje de programación Javascript. Se apoya en una librería externa llamada ExtJS, para el modelado de la página, su interfaz gráfica, y otra llamada OpenLayers para el renderizado de los mapas y las diferentes capas superpuestas.

4.2.2 Capa de Lógica de Negocio

En la parte **servidora,** se pretende integrar un servicio que proporcione los datos de tráfico en la plataforma Geoservicios, ya mencionada anteriormente. Los servicios web de esta plataforma originariamente estaban desarrollados con Visual Basic .NET, pero por otra parte la tendencia de la empresa era migrar toda esa funcionalidad a un servidor LINUX, por lo que el lenguaje VB no es compatible. Se tenía que elegir entre mantener la estructura anterior o crear el servicio web utilizando JAVA, que finalmente fue la opción elegida. Lo que si se quiso mantener por temas de compatibilidad con aplicaciones que utilizaban la anterior plataforma

fue el protocolo SOAP[2], por lo que el servicio está desarrollado con JAVA siguiendo el protocolo SOAP.

Para facilitar la comunicación entre el servicio web y el cliente (el navegador), se define un servicio *proxy* que hace de pasarela entre ambas partes. Esto es debido a que para la parte cliente, al ser Javascript, es mucho más ligero y rápido trabajar con JSON[3] en lugar de XML. Para mantener un mismo lenguaje de programación en toda la parte servidora, este servicio está desarrollado con JAVA, Servlet[4], y su respuesta es de tipo JSON.

La actualización de los datos en la base de datos, se realiza mediante un proceso en la parte servidora. Este proceso también se desarrollará con el mismo lenguaje de programación que los demás componentes de la parte servidora. Este proceso debe ser automático y se debe lanzar cada tres minutos, para ello se utiliza el mecanismo de Linux para procesos automáticos, CRON[5].

4.2.3 Capa de Acceso a Datos

En la gestión de datos, el requisito más importante que debe cumplir el sistema gestor, es que sea potente con los datos espaciales, es decir, los datos relacionados a los espacios en el mundo físico, las geometrías. Los tres sistemas de gestión de bases de datos más potentes para el almacenamiento y manipulación de este tipo de datos, son Oracle, Postgres y SqlServer con sus respectivas extensiones espaciales. En la empresa se trabaja con todos ellos, ya que cada cliente apuesta por el más conveniente para su empresa, por lo que no se tenía claro cuál utilizar. Entre ellos, Oracle y SqlServer necesitan estar licenciados, y Postgres es opensource. Así, se decidió utilizar Postgres, no sólo por ser software libre, sino también por su gran cantidad de documentación y su potencia de proceso.

Por otro lado, se estudió el posible uso de SQLite, ya que para el procesado de incidencias de TomTom, ellos recomiendan utilizar dicho sistema. Después de realizar varias pruebas, se llegó a la conclusión de que para mapas pequeños SQLite era una buena opción, pero en mapas más grandes como puede ser del País Vasco o de España, el rendimiento descendía.

---

[2] **SOAP** (*Simple Object Access Protocol*) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. (Fuente: http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol)

[3] **JSON** (*JavaScript Object Notation)* es un formato ligero para el intercambio de datos.(Fuente: http://es.wikipedia.org/wiki/JSON)

[4] **SERVLET** aplicaciones Java que corren en un entorno de servidor web. Son persistentes, lo que significa que una vez que han sido iniciados, se mantienen en memoria y pueden satisfacer múltiples solicitudes. (Fuente: http://www.alegsa.com.ar/Dic/servlet.php)

[5] **CRON** es un administrador regular de procesos en segundo plano (*demonio*) que ejecuta procesos o guiones a intervalos regulares (por ejemplo, cada minuto, día, semana o mes). Los procesos que deben ejecutarse y la hora en la que deben hacerlo se especifican en el fichero `crontab`.(Fuente: http://usemoslinux.blogspot.com/2010/11/cron-crontab-explicados.html)

En esta tabla están recogidos algunos datos de las pruebas realizadas:

	SQLite	PostreSQL
454 localizaciones	3min 36seg	2min 15seg
348 localizaciones	2min 52seg	1min 38seg
237 localizaciones	2min 14seg	1min 5seg
42 localizaciones	28seg	15seg

4.3 Arquitectura MVC sistema "Tráfico en Tiempo Real"

Se presenta la arquitectura general del sistema con las tecnologías utilizadas en cada nivel.

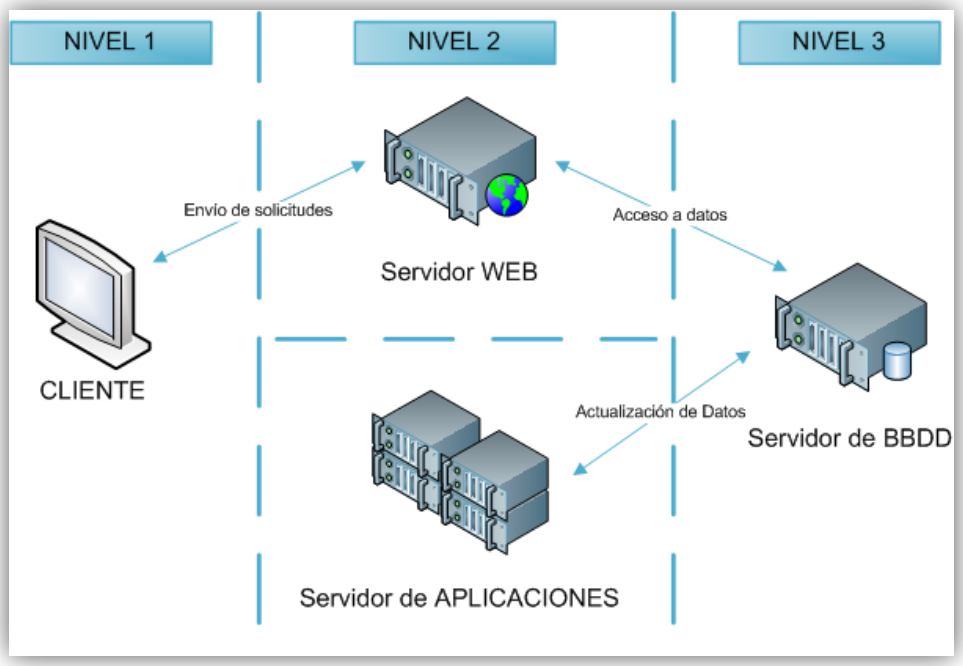


Diagrama 12: Arquitectura MVC sistema "Tráfico en Tiempo Real"

4.3.1 NIVEL 1: Capa de Presentación

Este nivel se ejecutará en el navegador web, y es el encargado de la interfaz de usuario, aunque también se encargará de la lógica de negocio a un nivel más bajo, solamente para la presentación de la información y la iteración con el usuario.

Su principal misión es la de enviar solicitudes a la capa de negocio, la relacionada con los servicios web, para la obtención de las incidencias, y la de mostrar dicha información sobre el navegador web favorito del usuario.

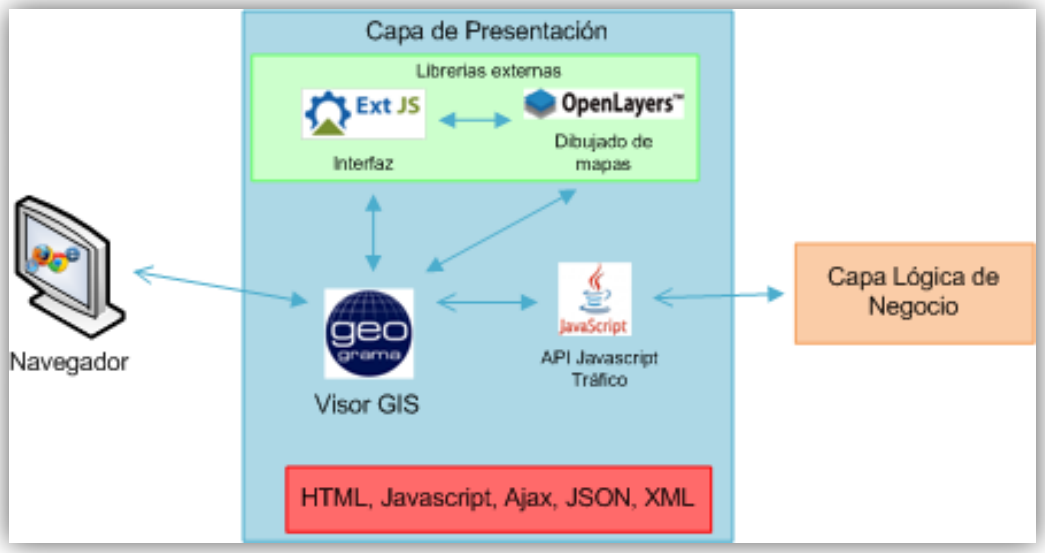


Diagrama 13: Capa de presentación

4.3.2 NIVEL 2: Capa de Lógica de Negocio

La lógica de negocio la realizará prácticamente en su totalidad el servidor. Estarán desplegados el servicio web SOAP y el Servlet utilizado como proxy, a parte del sistema que se dedicará al parseo, la decodificación y la actualización en la base de datos de las incidencias.

Por lo que este nivel está separado en dos capas, la primera para el consumo de los datos y la segunda para el proceso de dichos datos. Estas capas serán totalmente independientes entre ellas.

El Servlet desplegado en la primera de las dos capas de este nivel, se mantendrá a la escucha esperando alguna solicitud de la capa de presentación. En cuanto reciba una petición, delegará en el servicio web SOAP el acceso a los datos. Una vez el servicio web le responda, este convertirá la información obtenida en formato XML a formato JSON para enviársela a la capa de presentación.

El sistema de parseo, es el encargado de recoger el XML de incidencias proporcionado por TomTom, decodificar la ruta o el tramo de la incidencia, crear un modelo de clases y actualizar la base de datos con las nuevas incidencias eliminando las anteriores. Para la decodificación será necesario un mapa en formato BBDD y una librería opensource creada por TomTom llamada OpenLR.

Por lo que este nivel se tendrá que comunicar con los otros dos niveles de la arquitectura, la capa de presentación y la de acceso a datos, aunque la comunicación estará dividida independientemente entre los dos servicios de este nivel.

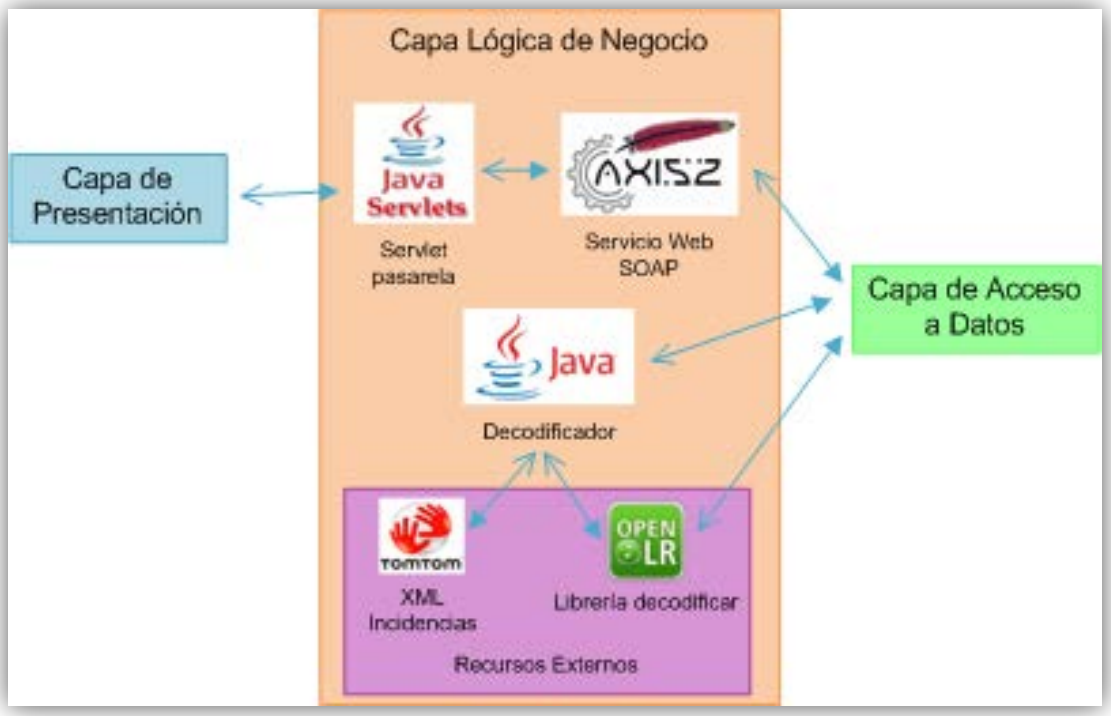


Diagrama 14: Capa lógica de negocio

4.3.3 NIVEL 3: Capa de Acceso a Datos

En nuestro sistema es imprescindible este nivel, ya que el almacenamiento de las incidencias es un punto muy importante. Este nivel se encargará de la creación de datos, lectura de datos, eliminación de datos y su modificación. Para ello se utiliza un sistema gestor de bases de datos, Postgres, junto con una extensión para la manipulación de los datos espaciales, PostGis.

Este nivel se comunicará con la capa de lógica de negocio mediante conexiones, aparte de realizar una pequeña parte de la lógica de negocio, para el tratamiento de los datos espaciales.

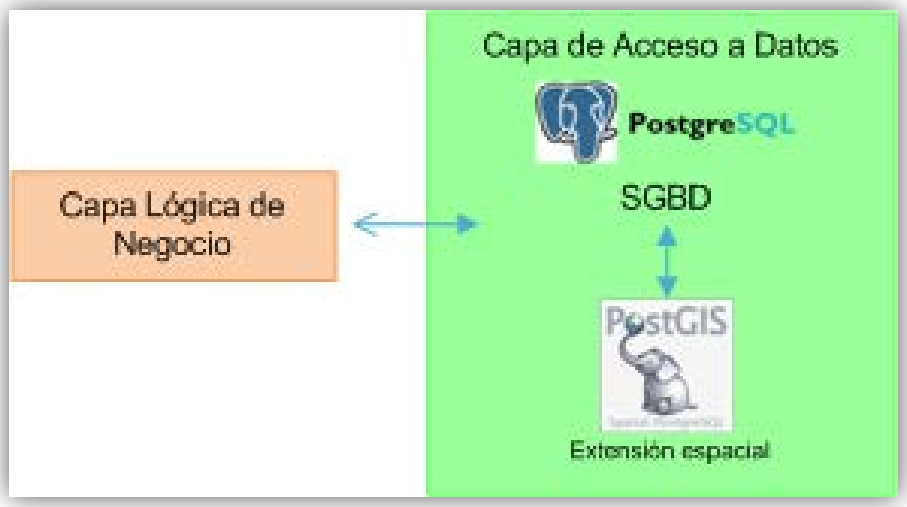


Diagrama 15: Capa de acceso a datos

4.4 Arquitectura Servidor

En este apartado incluimos un diagrama que nos muestra la estructura del servidor donde se van a desplegar tanto los servicios como la aplicación web. A grandes rasgos se compone por tres máquinas, una para el Gestor de Base de datos, otra en la que se alojará todo lo estático, es decir, las páginas web, los ficheros estáticos, las imágenes…y por último una para la máquina dinámica donde se encuentran desplegados tanto los servicios como los servidores de mapas que dispone la empresa:

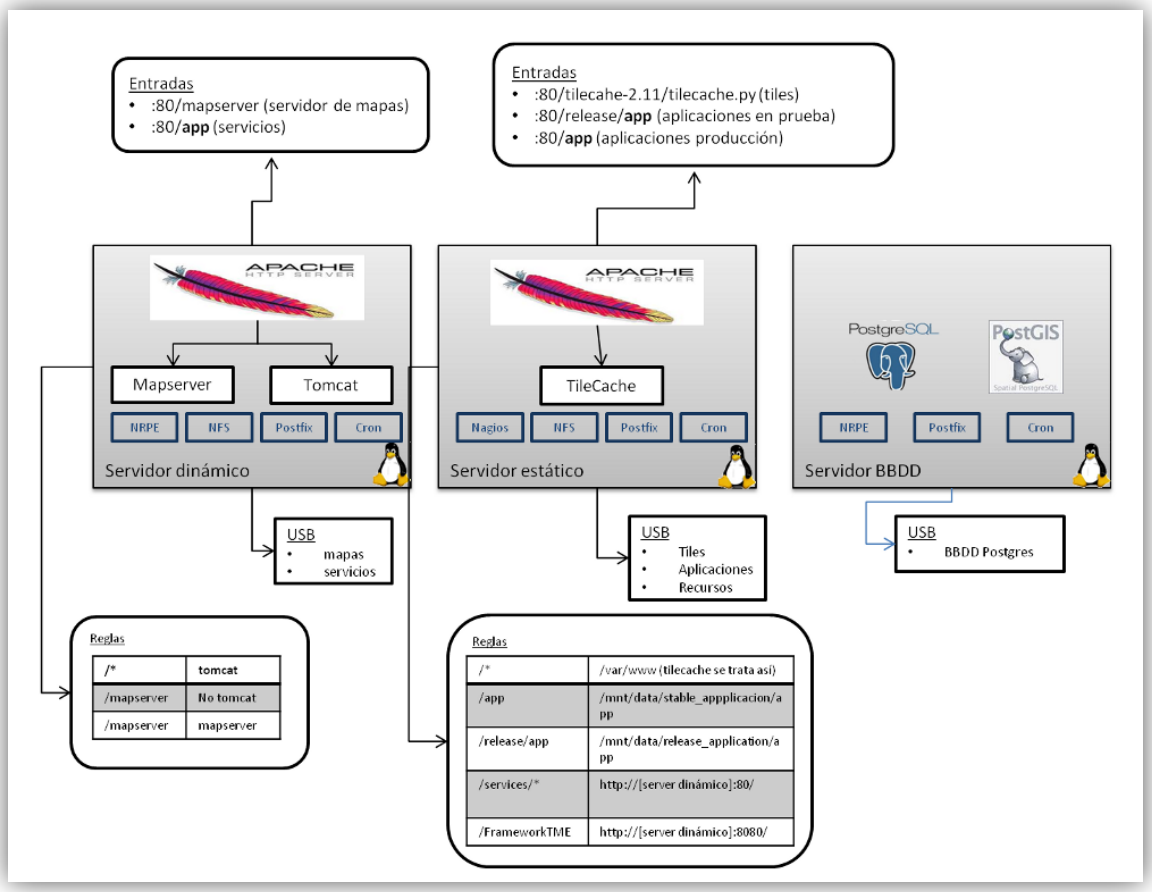


Figura 3: Arquitectura del Servidor

5. Diseño

5.1 Diseño de la interfaz

La interfaz gráfica de la aplicación de Tráfico en Tiempo Real sigue el modelo de los visores desplegados por la empresa. Consta de dos zonas importantes, el panel central donde se dibuja el mapa y herramientas sobre el mapa, y los paneles de la derecha o izquierda, donde se incluye algún tipo de funcionalidad sobre el mapa.

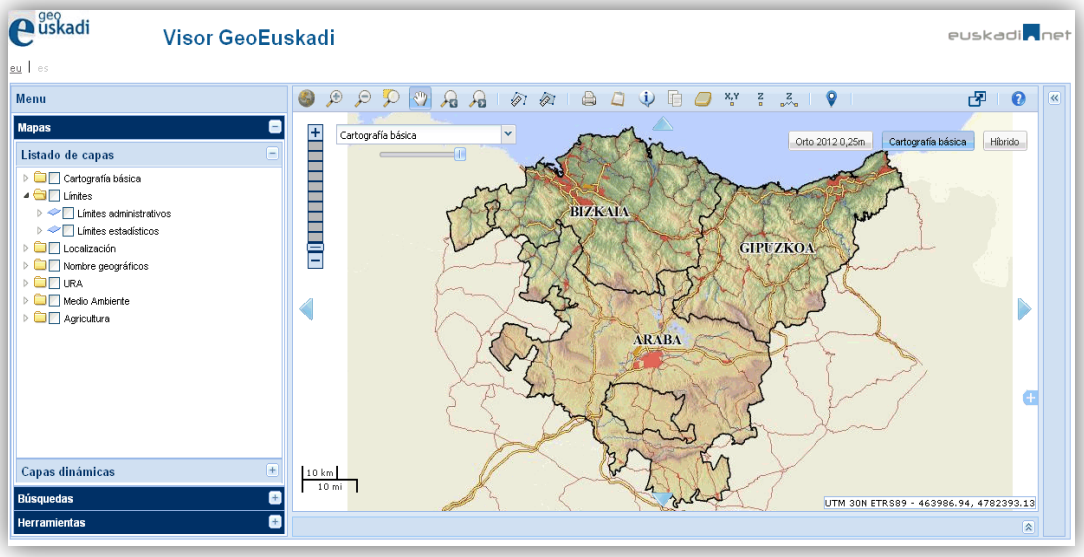


Figura 4: Ejemplo interfaz

Para las incidencias se han creado unos iconos que representan el inicio del tramo afectado y el tipo de incidencia, y se ha configurado el servidor de mapas para que dibuje una flecha al final del tramo.

Se ha creado un panel a la derecha con el listado de incidencias, y se incluye otro panel desarrollado anteriormente por la empresa, que permite activar y desactivar las diferentes capas de incidencias.

5.2 Diseño de Casos de Uso

A continuación se muestran en detalle todos los diagramas de secuencia asociados a los casos de uso que forman este proyecto web.

La sencillez de la mayoría de los diagramas viene dada por la simplificación en el código a la hora de realizar operaciones contra la base de datos utilizando procedimientos almacenados.

Para todos los diseños expuestos, he escogido el patrón controlador correspondiente para gestionar cada evento externo particular. Se selecciona un controlador de caso de uso para modelar cada caso de uso llamado *Gestor*. Con ello pretendemos un diseño global pero eficiente con alta cohesión y bajo acoplamiento.

5.2.1 Diseño de casos de uso del "Usuario"

5.2.1.1 Acceso al visor

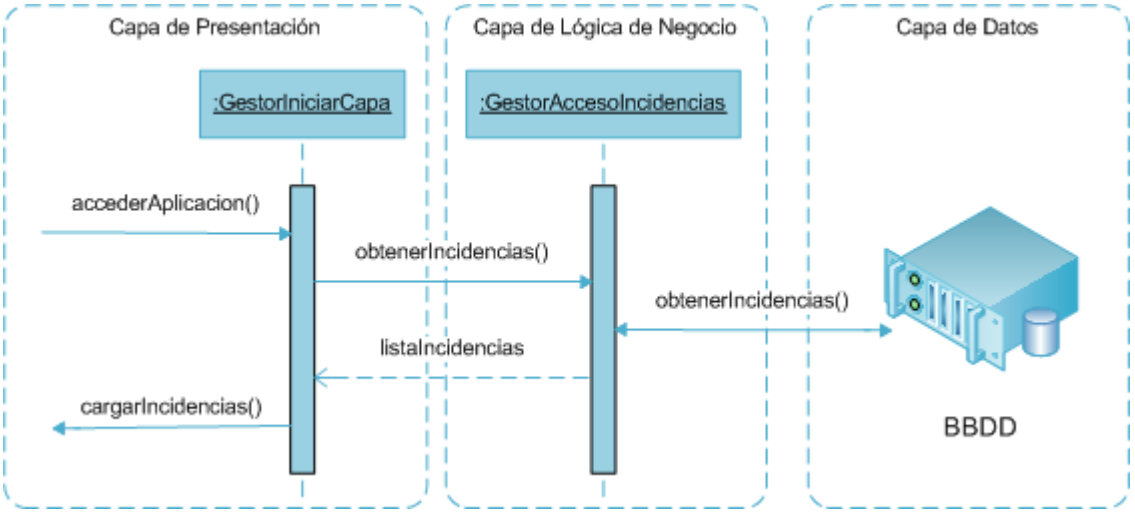


Diagrama 16: Diseño "Acceso al visor"

Seleccionamos el controlador de caso de uso :GestorIniciarCapa. El método accederAplicacion() inicializa la capa de incidencias, que debe acceder a los datos correspondientes con las incidencias almacenados en la BBDD, mediante el método obtenerIncidencias() y cargarlas en el mapa.

Por el patrón experto, el método obtenerIncidencias() es el encargado de acceder a la BBDD y recuperar el listado de incidencias, junto con toda su información asociada. El sistema dibujará en el mapa y cargará en el panel de información, mediante el método cargarIncidencias(), todas las incidencias obtenidas.

5.2.1.2 Ver Información

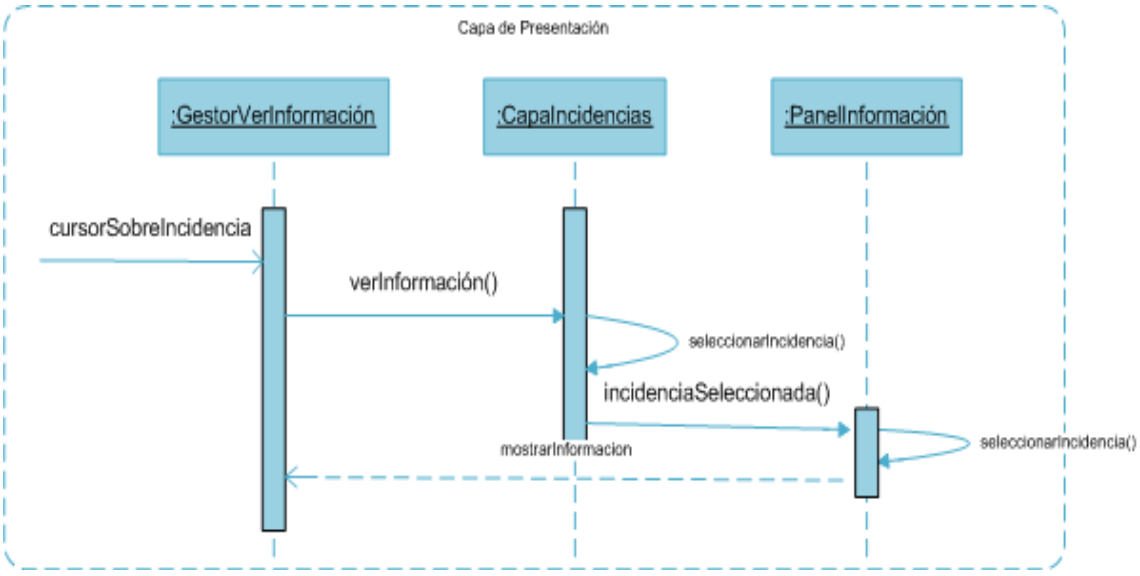


Diagrama 17: Diseño "Ver información"

Seleccionamos el controlador de caso de uso :GestorVerInformación. Cuando el usuario coloque el cursor sobre una incidencia en el mapa, se iniciará una serie de eventos mediante el método verInformación(). La capa de incidencias resaltará de algún modo dicho elemento.

A su vez, el método incidenciaSeleccionada() avisará al panel de información de que una incidencia ha sido seleccionada. Este mostrará la información asociada en el panel mediante el método mostrarInformación().

5.2.1.3 Seleccionar Incidencia

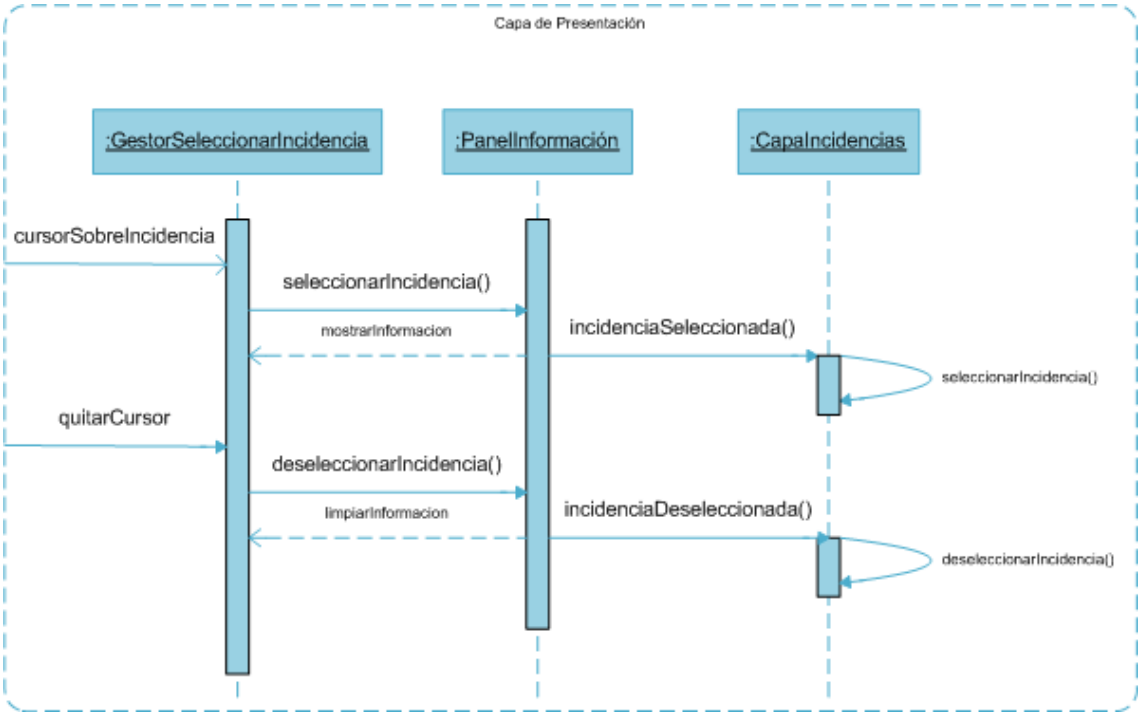


Diagrama 18: Diseño "Seleccionar Incidencia"

Seleccionamos el controlador de caso de uso :GestorSelecionarIncidencia. Cuando el usuario coloque el cursor sobre una incidencia en el panel de información de incidencias, se iniciará una serie de eventos mediante el método seleccionarIncidencia(). El panel seleccionará la incidencia, mostrando su información asociada en su parte inferior.

A su vez, el método incidenciaSeleccionada() avisará a la capa de incidencias de que una incidencia ha sido seleccionada. Esta resaltará el icono asociado a la selección sobre el mapa mediante el método seleccionarIncidencia() de la capa.

Mediante el controlador de caso de uso :GestorSeleccionarIncidencia, al quitar el cursor de la incidencia en el panel, se iniciará una serie de eventos mediante el método deseleccionarIncidencia(). El panel limpiará la información anteriormente mostrada quedándose en blanco.

El método incidenciaDeseleccionada(), se encargará de avisar a la capa de que la incidencia ha sido deseleccionada. En ese momento el icono de la incidencia volverá al tamaño inicial mediante el método deseleccionarIncidencia() de la capa.

5.2.1.4 Zoom a incidencia

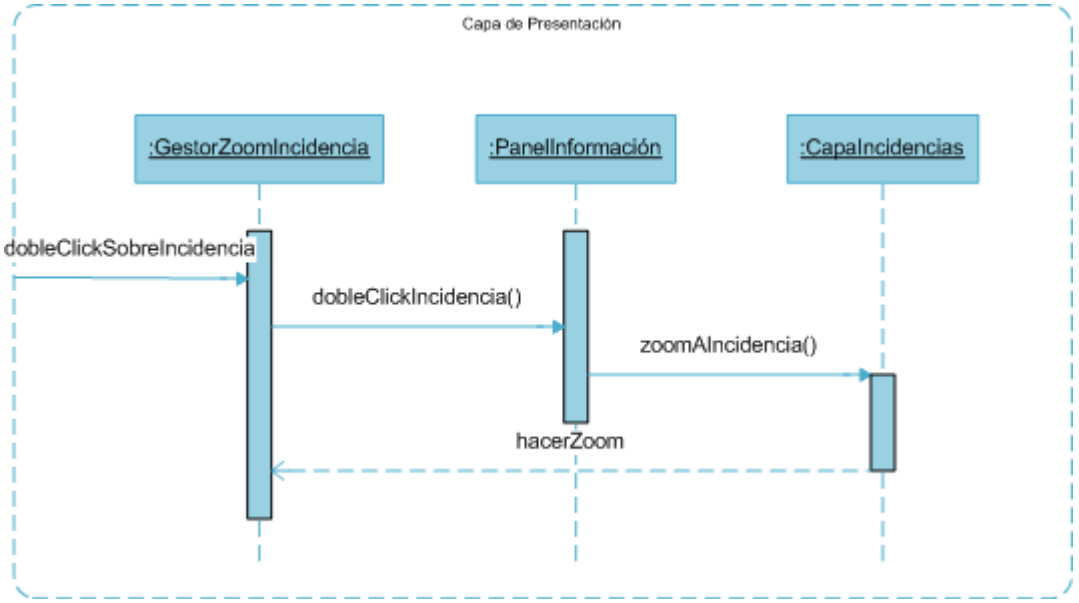


Diagrama 19: Diseño "Zoom a incidencia"

Seleccionamos el controlador de caso de uso :GestorZoomIncidencia. Cuando el usuario haga doble clic sobre una incidencia del panel de información, se llamará al método dobleClickIncidencia() que lanzará un evento.

La capa de incidencias capturará el evento mediante el método incidenciaSeleccionada(), De esta forma la capa de incidencias recogerá la información espacial o geográfica de la incidencia y se realizará un zoom en el mapa sobre la zona de la incidencia en cuestión.

5.2.2 Diseño de casos de uso del "Tiempo"

5.2.2.1 Refrescar visor

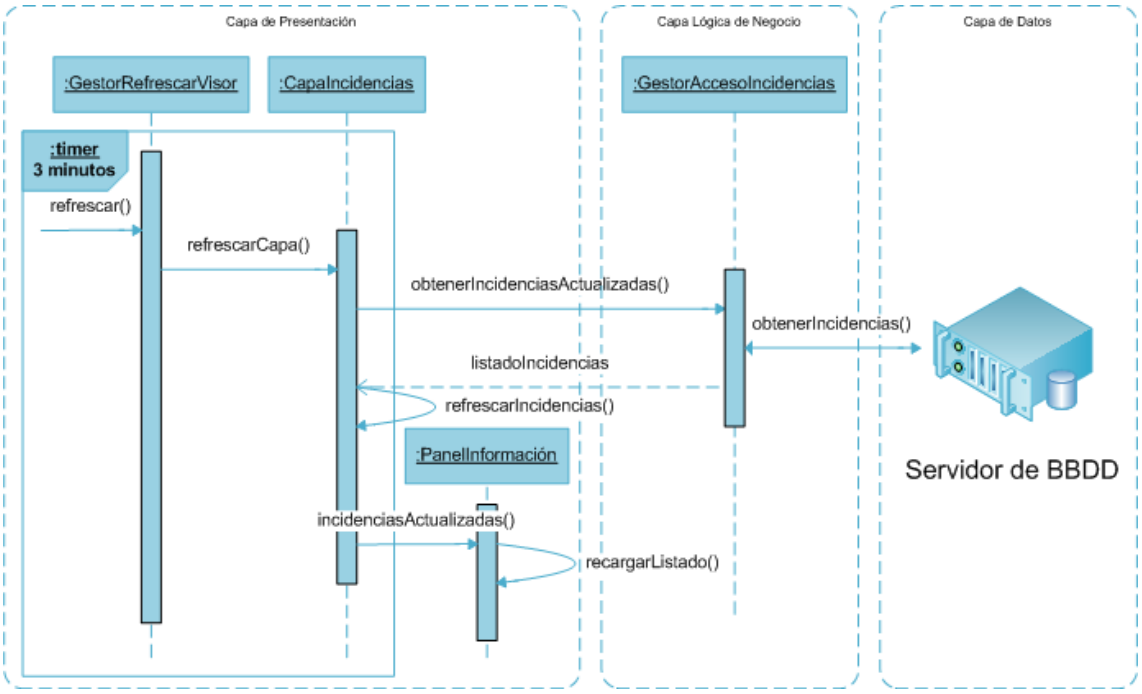


Diagrama 20: Diseño "Refrescar visor"

Seleccionamos el controlador de caso de uso :GestorRefrescarVisor. El controlador lanzará un temporizador que ejecutará la función refrescar() cada tres minutos sobre la capa de incidencias. Mediante la función refrescarCapa() la capa comenzará la traza para refrescar sus incidencias.

Por el patrón experto, el método obtenerIncidenciasActualizadas() es el encargado de comunicarse con la capa Lógica de Negocio que a su vez, se conectará con la base de datos para obtener el listado de incidencias actualizado junto a toda su información correspondiente, mediante la función obtenerInciencias().

Una vez el controlador de casos de uso :GestorObtenerIncidencias de la Lógica de Negocio, devuelva el listado, la capa se actualizará mediante el método refrescarIncidencias(). Cuando la capa finalice el refresco, avisará al panel de información de que se han actualizado las incidencias, mediante el método incidenciasActualizadas(), para que el panel rellene el listado.

Mediante el método recargarListado() el panel de información vaciará el listado anterior y lo recargará con el listado actualizado de incidencias.

5.2.2.2 Actualizar Datos

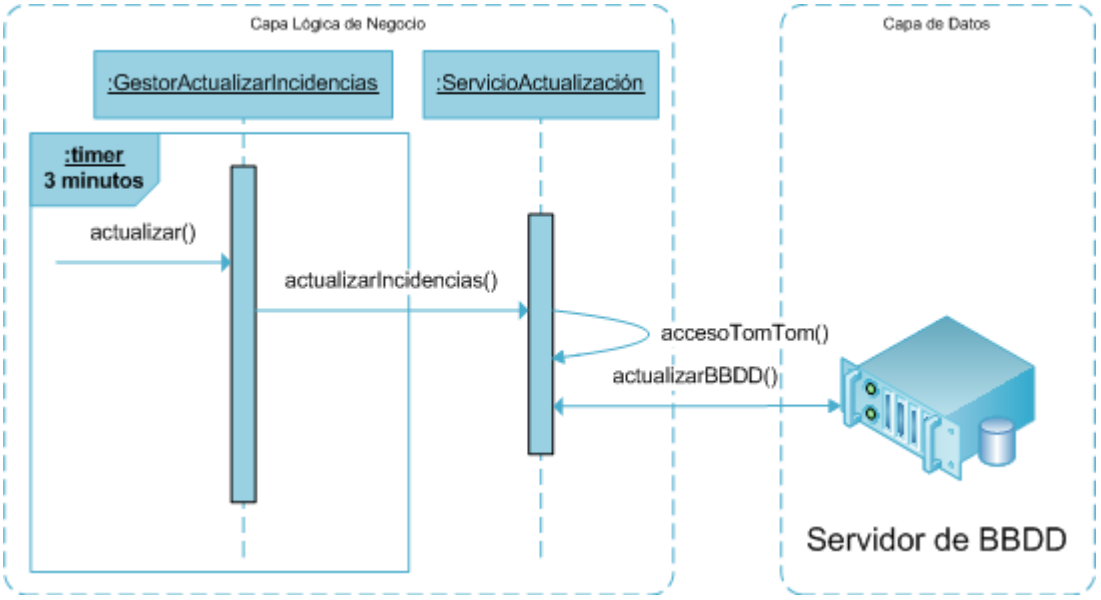


Diagrama 21: Diseño "Actualizar datos"

Seleccionamos el controlador de caso de uso :GestorActualizarIncidencias. El controlador lanzará un temporizador que ejecutará la función actualizar() cada tres minutos. Mediante la función actualizarIncidencias() el servicio de actualización de incidencias, comenzará la ejecución para actualizar las incidencias en la BBDD.

Por el patrón experto, el método accesoTomTom(), se conectará con el documento XML proporcionado por TomTom, y se descargará el listado de incidencias actuales. A continuación, mediante el método actualizarBBDD(), actualizará la tabla de incidencias almacenada en la base de datos, dejando de esa forma la información más reciente.

5.3 Diseño de la Base de Datos

El siguiente modelo de dominio representa las tablas asociadas al proyecto, donde se muestran las tablas necesarias para el almacenamiento y decodificación de las incidencias. Es un modelo de datos muy sencillo que cumple con las necesidades de la librería de decodificación OpenLR y de las necesidades propias del proyecto en sí.

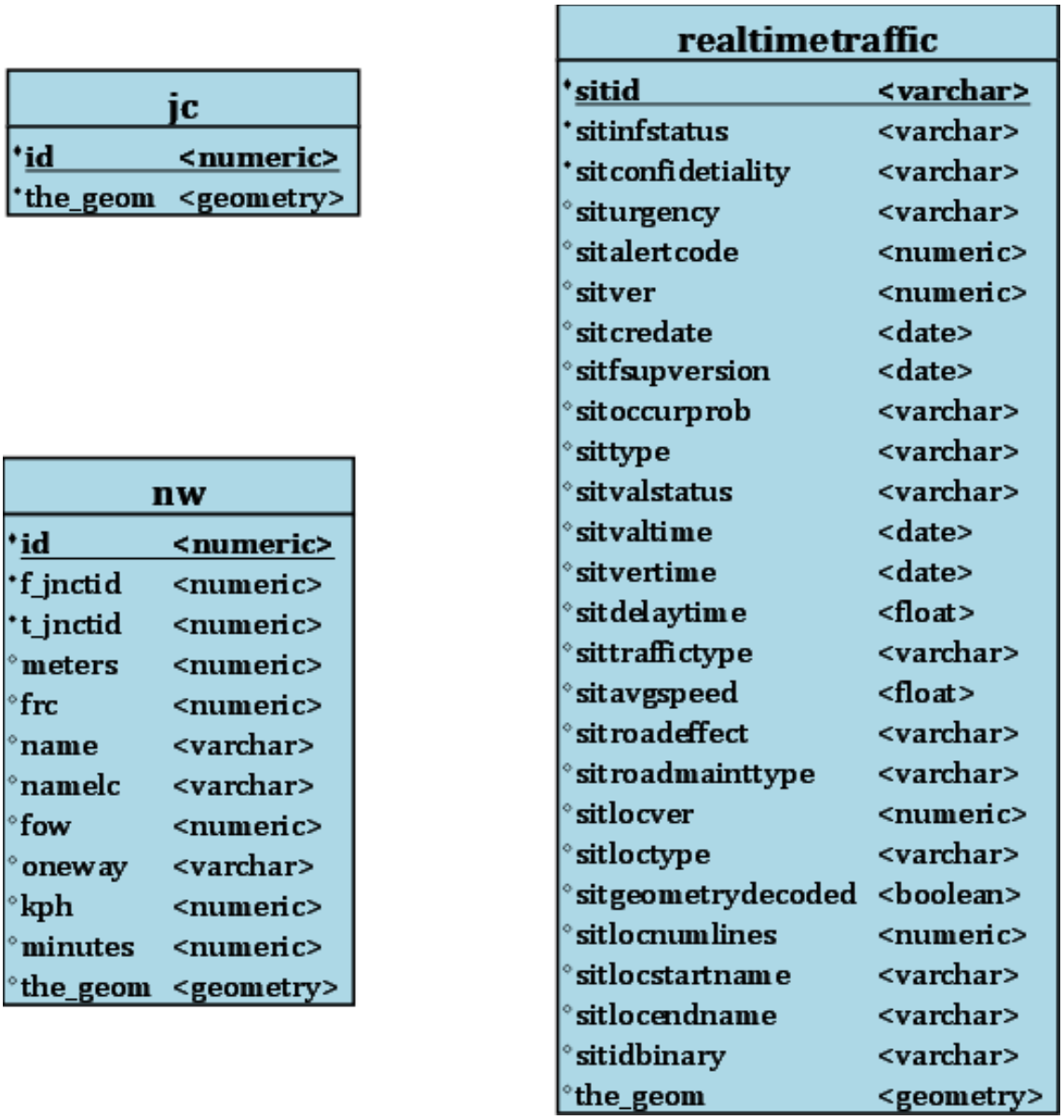


Diagrama 22: Diseño Base de Datos

5.3.1 Descripción general

En el modelo de dominio, se incluyen las tres tablas que toman parte en el sistema. Las dos tablas de la izquierda, *jc* y *nw*, están relacionadas con la decodificación de las incidencias, y la tabla *realtimetraffic* es la encargada de almacenar las incidencias obtenidas de TomTom.

5.3.1.1 Tablas "JC" y "NW"

Como ya se ha comentado en el apartado anterior, estas dos tablas se encargan de almacenar información necesaria para la decodificación de las incidencias. Estas tablas, son la representación modelada de un mapa de carreteras, en este caso del País Vasco. La tabla *nw* es el conjunto de tramos que componen la red de carreteras, con su geometría de tipo línea. A su vez *jc* es el listado de elementos puntuales o nodos, con su geometría asociada de tipo puntual, que componen los vértices de los tramos de *nw*, y de esa forma la unión entre los diferentes tramos.

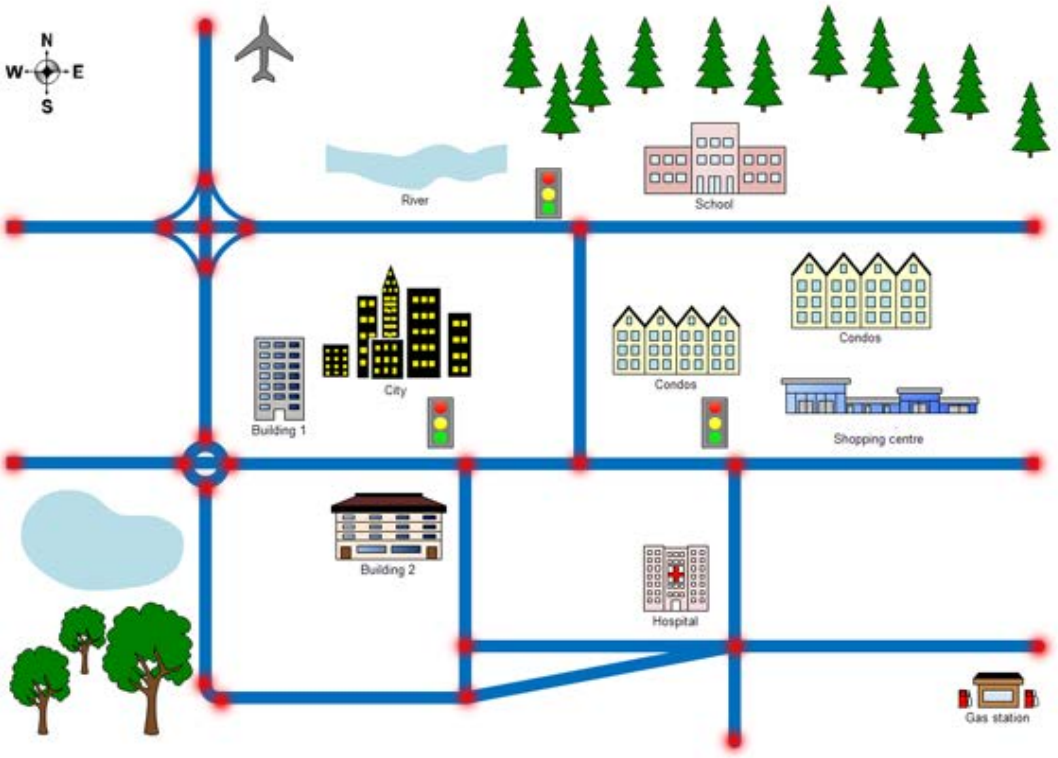


Ilustración 1: Ejemplo nodos y tramos

Donde cada punto rojo representa un registro de la tabla *jc*, y cada línea azul un registro de la tabla *nw*.

Descripción de los atributos de JC

Nombre	Tipo	Descripción
id	numeric	Identificador unívoco del nodo
the_geom	geometry	Representación espacial del nodo, de tipo punto (x,y)

Tabla 2: Atributos JC

Descripción de los atributos de NW

Nombre	Tipo	Descripción
id	numeric	Identificador unívoco del nodo.
f_jnctid	numeric	Identificador presente en la tabla jc que indica el nodo de inicio del tramo.
t_jnctid	numeric	Identificador presente en la tabla jc que indica el nodo fin del tramo.
meters	numeric	Longitud en metros del tramo.
frc	numeric	Tipo de carretera a la que pertenece el tramo: • 0: Autopistas o carreteras de más importancia. • 1: Autovías o carreteras similares. • 2: Carreteras nacionales o similares. • 3: Red secundaria de carreteras. • 4: Carreteras de conexiones locales. • 5: Carreteras locales o más importantes. • 6: Carreteras locales. • 7: Carreteras locales o de menos importancia. • 8: Otras vías.
name	varchar	Nombre oficial de la carretera a la que pertenece el tramo.
namelc	numeric	Código del idioma asociado al nombre del tramo.
fow	numeric	Forma del tramo: • 1: Parte de una autopista. • 2: Parte de carretera con varios carriles que no es una autopista. • 3: Parte de carretera de un único carril (default). • 4: Parte de una rotonda. • 6: Parte de un parking. • 7: Parte de un parking en un edificio. • 8: Parte de una plaza de tráfico • 10: Parte de tramo de acceso a autopista

		• 11: Parte de carretera de servicios • 12: Entrada / salida a un parking • 14: Parte de zona peatonal
oneway	varchar	Indica la dirección del tráfico: • Blank: Cerrada • FT: En dirección positiva a la geometría especificada • N: En ambas direcciones • TF: En dirección inversa a la geometría especificada
kph	numeric	Velocidad media calculada del tramo en km/h
minutes	numeric	Tiempo en recorrer este tramo en condiciones normales, en minutos.
the_geom	geometry	Representación espacial del tramo, una línea o un conjunto de líneas. $((x_1\ y_1, x_2\ y_2, \ldots, x_n\ y_n), (\ldots), (x_{1n}\ y_{1n}, x_{2n}\ y_{2n}, \ldots x_{nn}\ y_{nn}))$

Tabla 3: Atributos NW

5.3.1.2 Tabla "realtimetraffic"

En esta tabla se almacenan todas las incidencias recibidas desde el servicio de tráfico en tiempo real de TomTom. En ella se guarda información relacionada con el tipo de incidencia, la carretera a la que afecta, y la localización de dicha carretera.

Descripción de los atributos de realtimetraffic

Nombre	Tipo	Descripción
sitid	varchar	Identificador unívoco de la incidencia.
sitinfstatus	varchar	El estado de la información relacionada a la incidencia. (real, test, exercise ...)
sitconfidentiality	varchar	El grado de confidencialidad de la incidencia. (internalUse, noRrestriction, authoritiesOnly ...)
siturgency	varchar	Urgencia con la que el proveedor debe distribuir la incidencia.
sitalertcode	numeric	No definido.
sitver	numeric	Versión de la incidencia. Número de veces que ha ocurrido anteriormente.
sitcredate	date	Fecha y hora de alta de la incidencia.

sitfsupversion	date	Fecha y hora de alta de la incidencia.
sitoccurprob	varchar	Una evaluación del grado de probabilidad de que ocurra el evento reportado.
sittype	varchar	Tipo de incidencia. (AbnormalTraffic, MaintenanceWorks)
sitvalstatus	varchar	Validez de la incidencia.
sitvertime	date	No definido.
sitvertime	date	No definido.
sitdelaytime	float	Retraso en segundos que provoca la incidencia con respecto al tiempo normal de circulación del tramo.
sittraffictype	varchar	Tipo de retención de tráfico. (slowTraffic, stationaryTraffic, roadClosed…)
sitavgspeed	float	Velocidad de circulación media en el tramo afectado.
sitroadeffect	varchar	Efecto en la calzada debido a la incidencia.
sitroadmainttype	varchar	Tipo de obras en la calzada.
sitlocversion	numeric	No definido.
sitloctype	varchar	Tipo de geometría asociada a la localización de la incidencia. (point, linear)
sitgeometrydecoded	boolean	Indica si tiene una geometría asociada.
sitlocnumlines	varchar	Número de tramos de la tabla nw que se ven afectados por esta incidencia.
sitlocstartname	varchar	Nombre asociado al tramo inicial de la incidencia.
sitlocendname	varchar	Nombre asociado al último tramo asociado a la incidencia.
sitidbinary	varchar	Código binario que se debe decodificar para obtener la localización de la incidencia.
the_geom	geometry	Representación espacial de los tramos afectados por la incidencia. (Puntual, Multilinea o vacío)

Tabla 4: Atributos "realtimetraffic"

6. Implementación

En esta sección se explicarán los procedimientos seguidos para la implantación del sistema de Tráfico en Tiempo Real. Primero se mostrará la creación de la BBDD, a continuación se explicará todo el proceso de recogida de incidencias, decodificación y almacenamiento en la BBDD. Después se especificará el acceso a dichos datos en un modelo de tres niveles, y finalmente el consumo de esa información mediante el visor web.

En cada implementación tanto de las librerías como de los servicios, se ha definido un log, mediante la librería Log4java[6], que nos podrá servir de ayuda para monitorizar los errores controlados que se puedan producir durante el proceso. Más información en el Anexo I.

Como el código utilizado es muy extenso, en este documente solo se mostrará la información más relevante de cada parte.

6.1 Creación de la BBDD

La base de datos se compone de tres tablas principales, las relacionadas con la decodificación y con el almacenamiento de las incidencias. Para la creación de las dos primeras tablas, se utilizarán unos ficheros de datos espaciales llamados Shapefiles[7] proporcionados por TomTom. Como ya se ha comentado anteriormente, utilizaremos PostgreSql junto con su extensión para datos espaciales PostGis. Esta extensión nos proporciona una herramienta para importar un fichero shapefile a la base de datos desde la línea de comandos, llamada shp2pgsql. A continuación se incluye un .bat para crear las tablas mencionadas, mediante una serie de 9 shapefiles que contienen distinta información sobre los tramos y nodos que añadiremos a las tablas:

```
@echo off

set inicio=%date% %time%

cd espes1

shp2pgsql -DI -s 4326 -W UTF-8 espes1_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -DI -s 4326 -W UTF-8 espes1_____________nw nw | psql
nw_jc_spain postgres

shp2pgsql -Dn -s 4326 -W UTF-8 espes1_____________jcea jcea | psql
nw_jc_spain postgres
```

[6] **Log4java** es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o "logs" (data logging) a tiempo de ejecución y no a tiempo de compilación como es comúnmente realizado. (Fuente: http://es.wikipedia.org/wiki/Log4j)

[7] El formato **ESRI Shapefile** (SHP) es un formato de archivo informático propietario de datos espaciales desarrollado por la compañía ESRI, Actualmente se ha convertido en formato estándar de facto para el intercambio de información geográfica entre Sistemas de Información Geográfica. (Fuente: http://es.wikipedia.org/wiki/Shapefile)

```
shp2pgsql -Dn -s 4326 -W UTF-8 espes1_____________nwea nwea | psql
nw_jc_spain postgres

cd..
cd espes2

shp2pgsql -Da -s 4326 -W UTF-8 espes2_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes2_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes4


shp2pgsql -Da -s 4326 -W UTF-8 espes4_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes4_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes5

shp2pgsql -Da -s 4326 -W UTF-8 espes5_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes5_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes6

shp2pgsql -Da -s 4326 -W UTF-8 espes6_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes6_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes7

shp2pgsql -Da -s 4326 -W UTF-8 espes7_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes7_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes8

shp2pgsql -Da -s 4326 -W UTF-8 espes8_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes8_____________nw nw | psql
nw_jc_spain postgres

cd..
cd espes9

shp2pgsql -Da -s 4326 -W UTF-8 espes9_____________jc jc | psql
nw_jc_spain postgres
shp2pgsql -Da -s 4326 -W UTF-8 espes9_____________nw nw | psql
nw_jc_spain postgres

cd..
```

```
echo Proceso iniciado: %inicio%
echo Proceso finalizado: %date% %time%
pause
```

Una vez creadas las tablas, nos encontramos con un problema, que la información importada desde el shapefile no era compatible con la librería que se encarga de decodificar las incidencias (OpenLR). Por ejemplo, en la base de datos teníamos tramos de doble dirección, pero la librería solo entendía de tramos con una única dirección. Por lo que se tuvieron que adaptar estas tablas. A continuación se muestran las sentencias sql utilizadas para adaptar los datos:

```
--Extracción de registros sin duplicados; para ello he ejecutado esta
consulta (limites de los datasets producen nodos duplicados):
CREATE TABLE jc2 AS SELECT DISTINCT ON (id) id, the_geom FROM jc;

--Actualizar nw dependiendo de la dirección (doble dirección
duplicamos líneas)

  --Creamos una tabla temporal nueva nw2
  INSERT INTO nw2 SELECT * FROM nw WHERE oneway isnull;

  UPDATE nw2 SET id=id+100000000000000, the_geom=st_reverse(the_geom),
oneway='FT', f_jnctid=t_jnctid, t_jnctid=f_jnctid;

  INSERT INTO nw (id,feattyp, ft, f_jnctid, f_jncttyp,
t_jnctid,t_jncttyp ,pj
,meters,frc,netclass,netbclass,net2class,name,namelc,
  sol ,nametyp ,charge ,shieldnum  ,rtetyp ,rtedir  ,rtedirvd
,procstat ,fow ,sliprd ,freeway ,backrd ,tollrd ,rdcond ,stubble ,
  privaterd ,constatus ,  oneway ,f_bp ,  t_bp ,  f_elev ,  t_elev ,
kph ,  minutes ,  posaccur ,  carriage ,  lanes ,  ramp ,
  ada ,  trans ,  dynspeed ,  speedcat ,  nthrutraf ,  roughrd ,
partstruc ,  the_geom)
  SELECT id,feattyp, ft, f_jnctid, f_jncttyp, t_jnctid,  t_jncttyp ,
pj ,  meters,  frc,  netclass,  netbclass,  net2class,
  name,  namelc,  sol ,  nametyp ,  charge ,  shieldnum  ,  rtetyp ,
rtedir  ,  rtedirvd ,  procstat ,  fow ,  sliprd ,  freeway ,  backrd
,
  tollrd ,  rdcond ,stubble ,  privaterd ,  constatus ,  oneway ,
f_bp ,  t_bp ,  f_elev ,  t_elev ,  kph ,  minutes ,  posaccur ,
carriage ,
  lanes ,ramp ,  ada ,  trans ,  dynspeed ,  speedcat ,  nthrutraf ,
roughrd ,  partstruc ,  the_geom FROM nw2;

  DROP TABLE nw2;

--Actualizar nw dependiendo de la dirección (dirección inversa)
UPDATE nw SET the_geom=st_reverse(the_geom), oneway='FT',
f_jnctid=t_jnctid, t_jnctid=f_jnctid where oneway='TF';


--Actualizar el campo FOW de nw para OpenLR:

UPDATE nw SET fow = 0 where fow = -1
UPDATE nw SET fow = 5 where fow = 8
UPDATE nw SET fow = 7 where (fow = 6 OR fow > 10)
UPDATE nw SET fow = 6 where fow = 10
```

```
--Actualizar el campo FRC de nw para OpenLR:
UPDATE nw SET frc = 7 where (frc = -1 OR frc = 8)
```

La tabla *realtimetraffic* es una tabla simple que se creará de la forma habitual utilizando una sentencia SQL.

6.2 Recuperación, decodificación y almacenamiento de incidencias

Este apartado, con el fin de hacerlo más entendible, lo dividiremos en dos. Por un lado se explicará la librería de decodificación creada "Decoder", y por otro lado la librería encargada de la recuperación y almacenamiento de las incidencias "Parser".

6.2.1 Librería "Decoder"

Esta librería, que la llamaremos "Decoder", se basa en el nuevo estándar de TomTom para la codificación, transmisión y decodificación de datos espaciales, llamado OpenLR. Para ello nuestra librería deberá implementar una interfaz, que se compone por una serie de clases escritas en java, ofrecida por este producto.

6.2.1.1 Configuración

En primer lugar generamos un fichero de configuración de la base de datos. Con este fichero, lo que se pretende es independizar la base de datos de la librería, y de esta forma poder configurar diferentes tipos de bases de datos para utilizarlos con la misma librería sin tener que modificar ni una sola línea de código.

Este fichero será un documento xml, en el que se indicarán los datos de conexión a la base de datos, los nombres de los campos de las tablas "jc" y "nw", y por último las consultas que se realizarán sobre la base de datos.

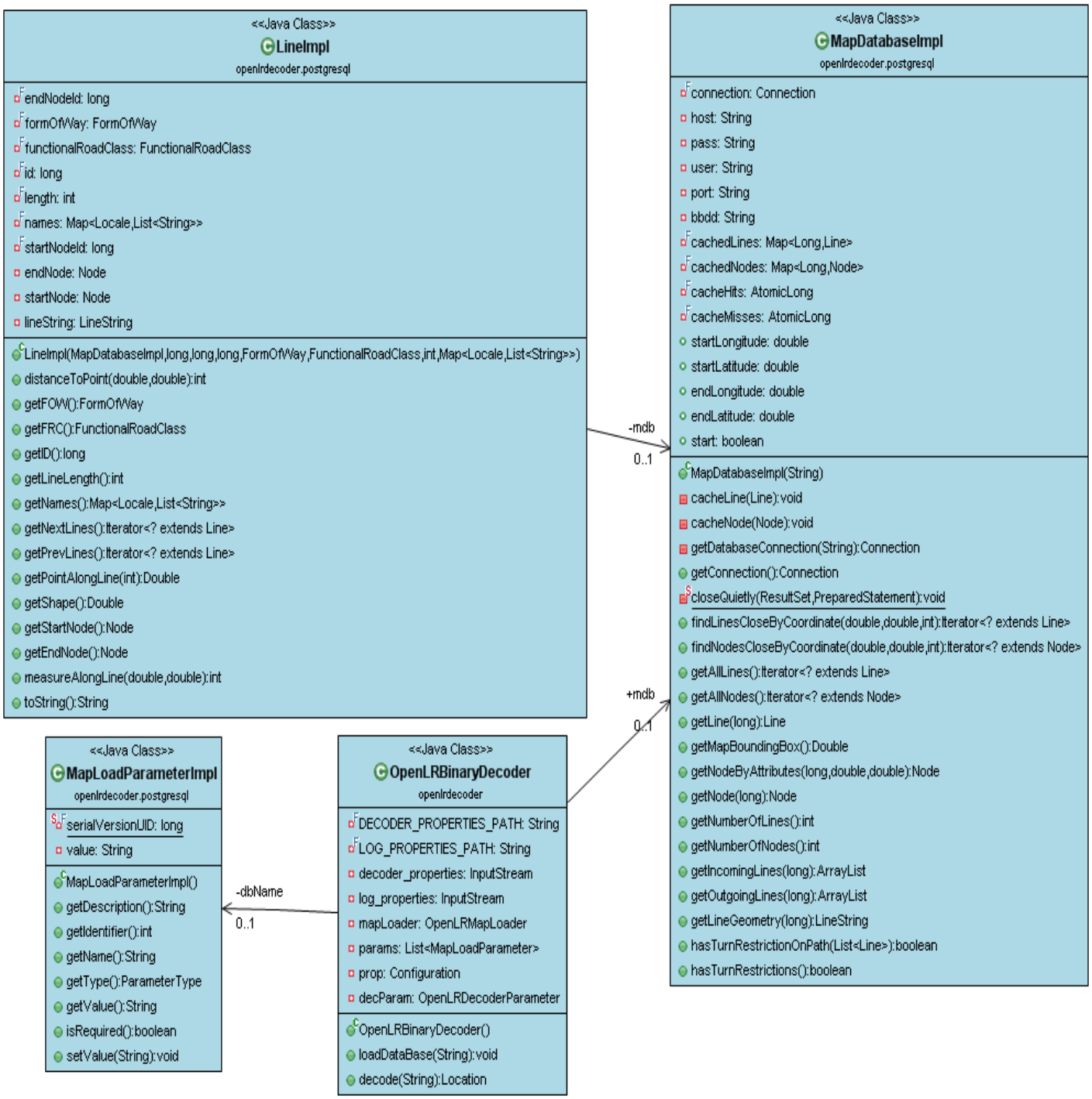
Este es un fragmento del fichero de configuración como demostración.

```
    <!-- Datos de conexión-->
    <entry key="port">5432</entry>
    <entry key="host">localhost</entry>
    <entry key="password">hen.7822</entry>
    <entry key="user">postgres</entry>

    <!-- Campos de "jc" -->
    <entry key="Node.Name">jc</entry>
    <entry key="Node.Column.Id">id</entry>
    <entry key="Node.Column.Longitude">lon</entry>
    <entry key="Node.Column.Latitude">lat</entry>
    <entry key="Node.Column.Geometry">the_geom</entry>

    <!-- Consulta que devuelve la extensión total de todos los nodos
    de "jc" -->
    <entry key="Node.Statement.BoundingBox">
            SELECT
                xmin(st_envelope({Node.Column.Geometry})) as xmin,
                xmax(st_envelope({Node.Column.Geometry})) as xmax,
                ymin(st_envelope({Node.Column.Geometry})) as ymin,
                ymax(st_envelope({Node.Column.Geometry})) as ymax,
        FROM
            {Node.Name}
```

```
    </entry>


    <!-- Consulta que dado un identificador de una tramo de "nw"
    devuelve la geometria en formato de texto asociada a dicho
    tramo -->
    <entry key="Line.Statement.Geometry">
        SELECT
            st_asewkb(st_geometryn({Line.Column.Geometry},1)) as
            {Line.Column.Geometry}
        FROM
            {Line.Name}
        WHERE
            {Line.Column.Id} = ?
    </entry>
```

6.2.1.2 Implementación de la interfaz

Se deberán generar una serie de clases que siguen una interfaz ofrecida por la librería OpenLR. A continuación se expone el diagrama de clases asociado a la librería de decodificación junto a una pequeña explicación.



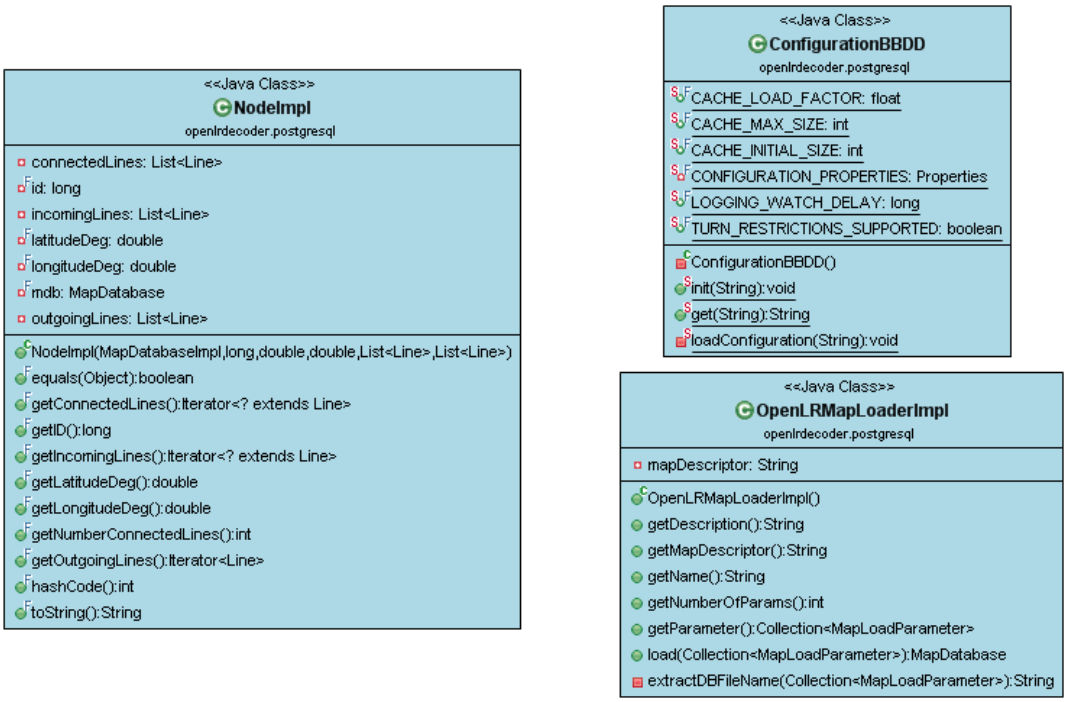


Diagrama 23: Diagrama de clases interfaz OpenLR

ConfigurationBBDD.java

Esta clase es la encargada de leer el fichero *Database.xml* especificado en el apartado anterior y guardar en memoria los diferentes apartados de dicho documento.

MapDatabaseImpl.java

Esta clase define una serie de métodos necesarios para la decodificación de las incidencias. Es la encargada de crear y cerrar la conexión con la base de datos, y se encarga de ejecutar las consultas almacenadas en la clase *ConfigurationBBDD.java.* A su vez define una caché en base a objetos *Map* nativos de java, para almacenar los nodos y tramos y de esa forma optimizar los accesos a la base de datos. Para información más detallada ver Anexo III.

OpenLRMapLoaderImpl.java

Se encarga de inicializar el mapa. Esta clase esta proporcionada por OpenLR por lo que no necesita desarrollo.

MapLoadParameterImpl.java

Se encarga de cargar los parámetros del mapa. Esta clase esta proporcionada por OpenLR por lo que no necesita desarrollo.

NodeImpl.java

Equivale a un nodo de la tabla “jc”. En esta clase se especifican el identificador del nodo, un listado de tramos que se inician en este nodo, un listado de tramos que finalizan con este nodo y finalmente las coordenadas x, y.

LineImpl.java

Equivale a un tramo de la tabla "nw". En esta clase se definen el identificador del tramo, el tipo de tramo, la longitud en metros, los nodos "nw" de inicio y final, el nombre asociado al tramo y finalmente su geometría en formato línea.

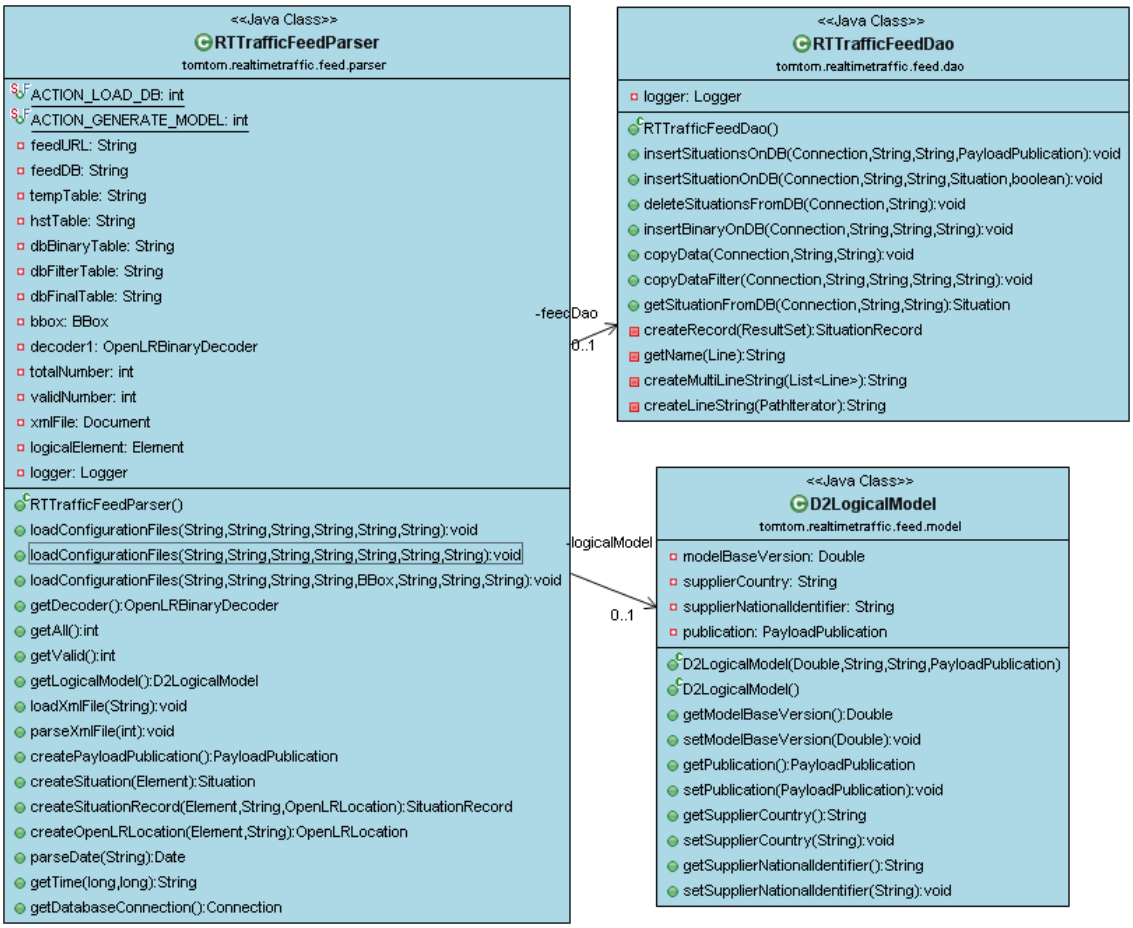
OpenLRBinaryDecoder.java

Esta clase es la única que no implementa ninguna interfaz de OpenLR. Se encarga de inicializar todas las clases necesarias, como OpenLRMapLoaderImpl y MapLoadParameterImpl. Contiene a su vez el método que se encarga de decodificar una incidencia dado un código binario como String "*decode(String binaryCode)".* La librería encargada de parsear el documento xml con el listado de incidencias utilizará esta clase para decodificar las incidencias. Para más información ver Anexo III.

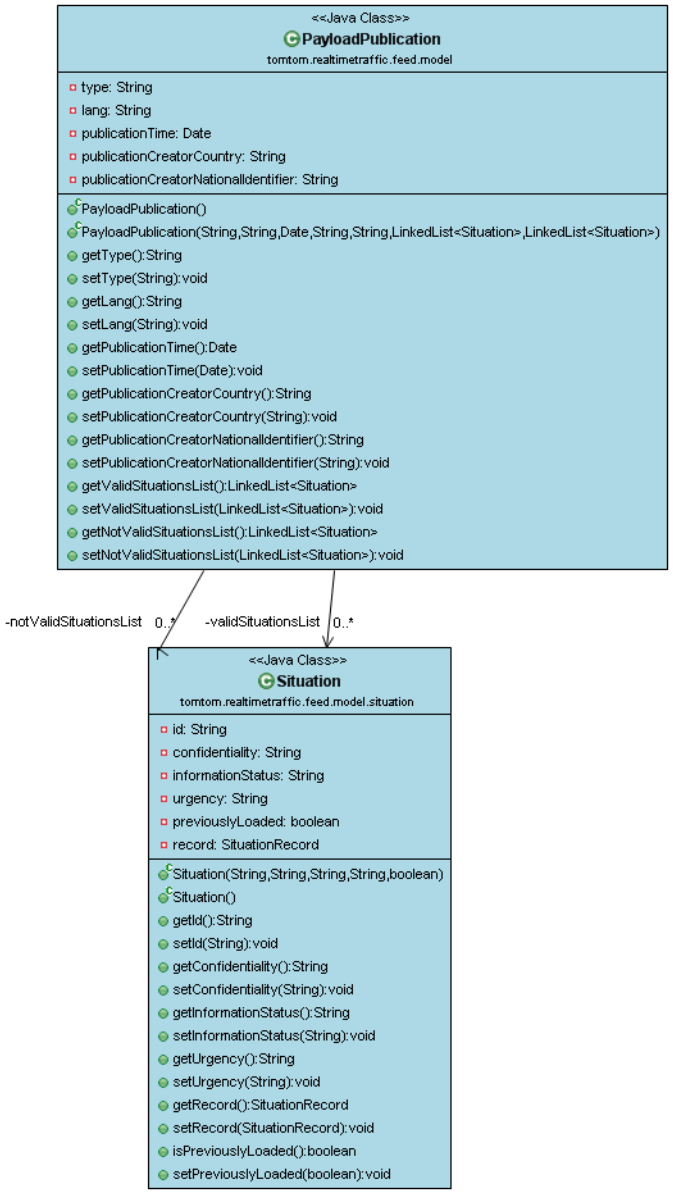
6.2.2 Librería "Parser"

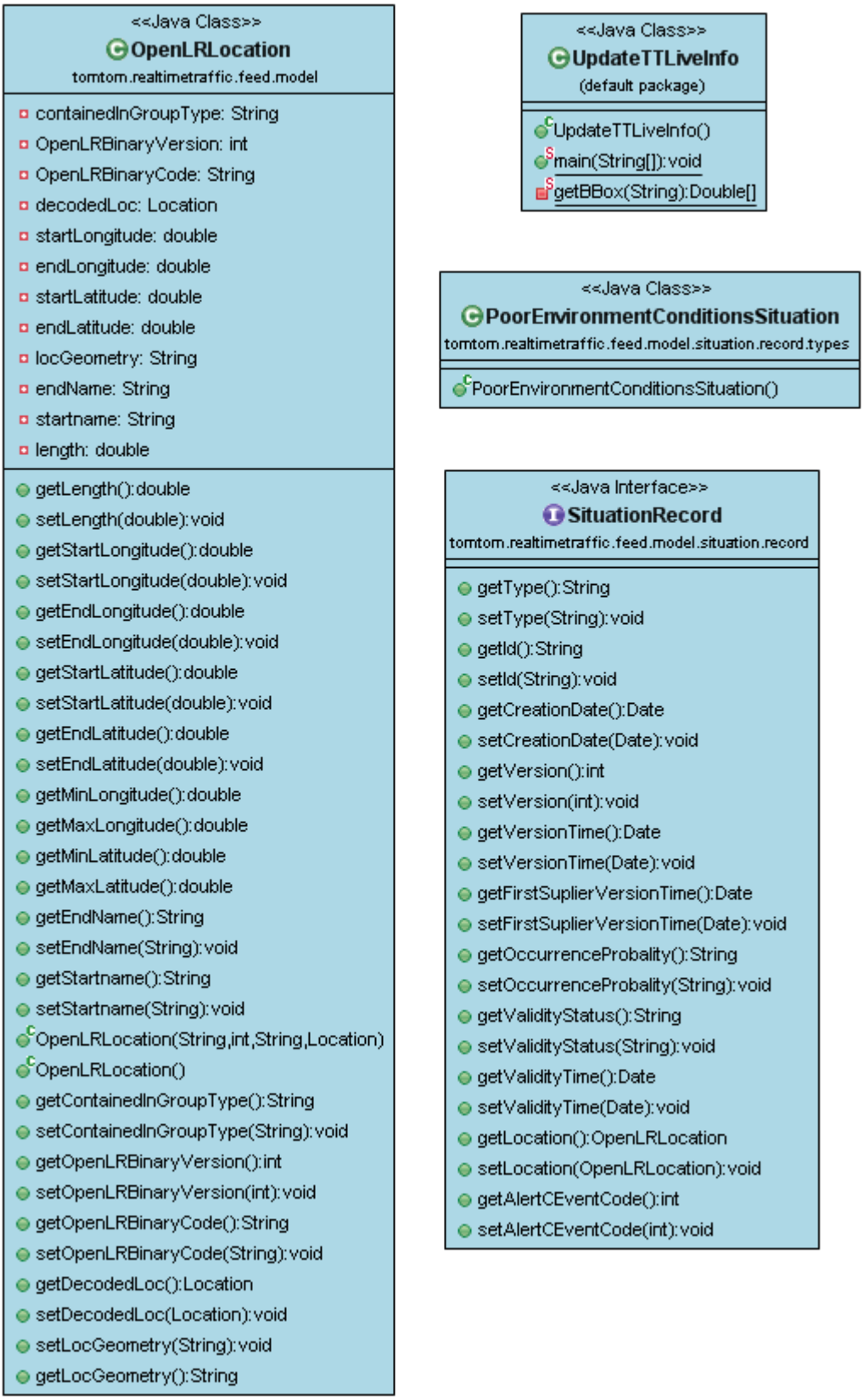
En este apartado, se explicará cómo funciona la librería creada para la recuperación de las incidencias, su decodificación mediante la librería indicada anteriormente, "Decoder", y finalmente su almacenamiento en la base de datos. En el Anexo IV se muestra la parte más significativa de su código y ejemplos.

Primero mostramos el diagrama de clases utilizado, como es un diagrama algo complicado, lo mostraremos en varios apartados:




<<Java Class>>
MaintenanceWorksSituation
tomtom.realtimetraffic.feed.model.situation.record.types
effectOnRoad: String
roadMaintenanceType: String
type: String
id: String
creationDate: Date
version: int
versionTime: Date
firstSuplierVersionTime: Date
occurrenceProbality: String
validityStatus: String
validityTime: Date
location: OpenLRLocation
alertCEventCode: int
MaintenanceWorksSituation(String,String,String,String,Date,int,Date,Date,String,String,Date,OpenLRLocation,int)
MaintenanceWorksSituation()
getEffectOnRoad():String
setEffectOnRoad(String):void
getRoadMaintenanceType():String
setRoadMaintenanceType(String):void
getType():String
setType(String):void
getId():String
setId(String):void
getCreationDate():Date
setCreationDate(Date):void
getVersion():int
setVersion(int):void
getVersionTime():Date
setVersionTime(Date):void
getFirstSuplierVersionTime():Date
setFirstSuplierVersionTime(Date):void
getOccurrenceProbality():String
setOccurrenceProbality(String):void
getValidityStatus():String
setValidityStatus(String):void
getValidityTime():Date
setValidityTime(Date):void
getLocation():OpenLRLocation
setLocation(OpenLRLocation):void
getAlertCEventCode():int
setAlertCEventCode(int):void


<<Java Class>>
PayloadPublication
tomtom.realtimetraffic.feed.model
type: String
lang: String
publicationTime: Date
publicationCreatorCountry: String
publicationCreatorNationalIdentifier: String
PayloadPublication()
PayloadPublication(String,String,Date,String,String,LinkedList<Situation>,LinkedList<Situation>)
getType():String
setType(String):void
getLang():String
setLang(String):void
getPublicationTime():Date
setPublicationTime(Date):void
getPublicationCreatorCountry():String
setPublicationCreatorCountry(String):void
getPublicationCreatorNationalIdentifier():String
setPublicationCreatorNationalIdentifier(String):void
getValidSituationsList():LinkedList<Situation>
setValidSituationsList(LinkedList<Situation>):void
getNotValidSituationsList():LinkedList<Situation>
setNotValidSituationsList(LinkedList<Situation>):void
-notValidSituationsList 0..*
-validSituationsList 0..*
<<Java Class>>
Situation
tomtom.realtimetraffic.feed.model.situation
id: String
confidentiality: String
informationStatus: String
urgency: String
previouslyLoaded: boolean
record: SituationRecord
Situation(String,String,String,String,boolean)
Situation()
getId():String
setId(String):void
getConfidentiality():String
setConfidentiality(String):void
getInformationStatus():String
setInformationStatus(String):void
getUrgency():String
setUrgency(String):void
getRecord():SituationRecord
setRecord(SituationRecord):void
isPreviouslyLoaded():boolean
setPreviouslyLoaded(boolean):void


<<Java Class>>
OpenLRLocation
tomtom.realtimetraffic.feed.model
containedInGroupType: String
OpenLRBinaryVersion: int
OpenLRBinaryCode: String
decodedLoc: Location
startLongitude: double
endLongitude: double
startLatitude: double
endLatitude: double
locGeometry: String
endName: String
startname: String
length: double
getLength():double
setLength(double):void
getStartLongitude():double
setStartLongitude(double):void
getEndLongitude():double
setEndLongitude(double):void
getStartLatitude():double
setStartLatitude(double):void
getEndLatitude():double
setEndLatitude(double):void
getMinLongitude():double
getMaxLongitude():double
getMinLatitude():double
getMaxLatitude():double
getEndName():String
setEndName(String):void
getStartname():String
setStartname(String):void
OpenLRLocation(String,int,String,Location)
OpenLRLocation()
getContainedInGroupType():String
setContainedInGroupType(String):void
getOpenLRBinaryVersion():int
setOpenLRBinaryVersion(int):void
getOpenLRBinaryCode():String
setOpenLRBinaryCode(String):void
getDecodedLoc():Location
setDecodedLoc(Location):void
setLocGeometry(String):void
getLocGeometry():String
<<Java Class>>
UpdateTTLiveInfo
(default package)
UpdateTTLiveInfo()
main(String[]):void
getBBox(String):Double[]
<<Java Class>>
PoorEnvironmentConditionsSituation
tomtom.realtimetraffic.feed.model.situation.record.types
PoorEnvironmentConditionsSituation()
<<Java Interface>>
SituationRecord
tomtom.realtimetraffic.feed.model.situation.record
getType():String
setType(String):void
getId():String
setId(String):void
getCreationDate():Date
setCreationDate(Date):void
getVersion():int
setVersion(int):void
getVersionTime():Date
setVersionTime(Date):void
getFirstSuplierVersionTime():Date
setFirstSuplierVersionTime(Date):void
getOccurrenceProbality():String
setOccurrenceProbality(String):void
getValidityStatus():String
setValidityStatus(String):void
getValidityTime():Date
setValidityTime(Date):void
getLocation():OpenLRLocation
setLocation(OpenLRLocation):void
getAlertCEventCode():int
setAlertCEventCode(int):void

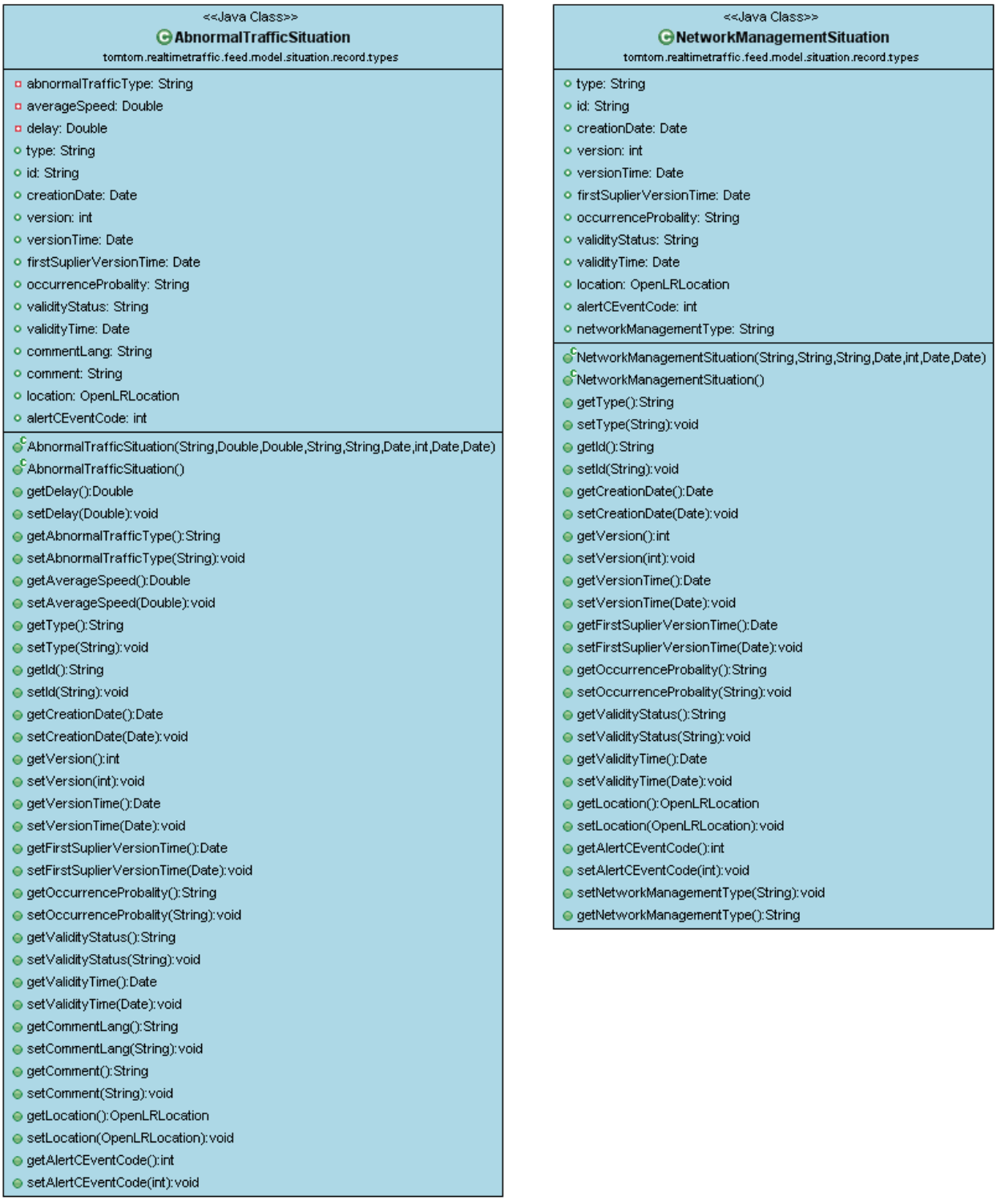


Diagrama 24: Diagrama de clases "Parser"

UpdateTTLiveInfo.java

La clase principal, la que se debe ejecutar cada 3 minutos para actualizar todo el listado de incidencias de la base de datos. Esta clase se inicializa mediante un CRON configurado en la máquina de Linux donde se encuentra desplegada la librería. Tiene un método principal que inicializa la clase

"RTTrafficFeedParser.java" con los parámetros necesarios como los datos de acceso a la BBDD y la url del servicio de incidencias de tráfico en tiempo real.

RTTrafficFeedParser.java

Como ya se ha explicado en el punto anterior, esta clase la instancia la clase principal de la librería, "UpdateTTLiveInfo.java". En primer lugar, se encarga de leer el documento xml proporcionado por TomTom (en el Anexo IV se puede ver un documento de ejemplo y su lectura). Una vez cargado en memoria el xml, se inicia el parseo del documento. Mediante las clases definidas dentro del paquete "tomtom.realtimetraffic.feed.model", se modeliza el documento y se almacena el listado de incidencias en el modelo de clases. En este punto, es donde se utilizará la librería "Decoder" para obtener la localización de la incidencia. Lo que se ha pretendido con este modelo de clases, es reflejar el documento lo más sencillo posible para su acceso directo mediante estas clases en Java.

D2LogicalModel.java

Es la clase principal del modelo de clases asociado al xml de incidencias. En esta clase se definen atributos generales del documento como la versión y el país desarrollador. Para nuestra librería estos datos no son relevantes pero se ha intentado modelar el documento completo.

PayloadPublication.java

En esta clase se indican varios atributos, generales también como en la clase anterior, como por ejemplo el idioma, y la fecha de publicación. Pero lo que realmente nos importa de esta clase son los dos listados de incidencias. Uno con las que se han conseguido decodificar y otro con las que no. Esto es debido a que en el documento están incluidas todas las incidencias de España, y puede ser que nuestro mapa, "jc" y "nw", solamente contemple una determinada zona, por lo que las que no se encuentren en esa zona no se podrán decodificar.

Situation.java

La clase que representa una incidencia. En ella están definidos los atributos más generales de cada incidencia.

SituationRecord.java <Interface>

Una *Interface* que define una serie de métodos para guardar y obtener las propiedades principales de la incidencia. Se ha definido una *Interface* a este nivel ya que las incidencias pueden ser de diferentes tipos, por lo que esos tipos que se explicarán a continuación, deberán implementar esta clase.

AbnormalTrafficSituation.java

Representa una incidencia de tipo tráfico, es decir, un atasco, tráfico lento o retención. Implementa la *Interface* "SituationRecord.java" comentada anteriormente.

MaintenanceWorksSituation.java

Representa una incidencia de tipo obras en la calzada. Implementa la *Interface* "SituationRecord.java" comentada anteriormente.

NetworkManagementSituation.java

Representa una incidencia de tipo gestión de la red de carreteras, por ejemplo, carril abierto en sentido contrario o carretera cortada. Implementa la *Interface* "SituationRecord.java" comentada anteriormente.

OpenLRLocation.java

Esta clase se encuentra referenciada en las clases que implementan la *Interface* "SituationRecord.java". En esta clase se especifican los atributos referentes a la geometría de la incidencia. Se indica el código binario que se debe decodificar, así como el tipo de geometría, el nombre asociado al tramo o punto de nuestro mapa, "jc" y "nw" o la geometría en cuestión.

6.3 Servicio en tres niveles

Para la obtención de las incidencias almacenadas en la base de datos, se han definido tres niveles diferenciados, servicio SOAP, servicio Servlet y api Javascript. Esto se plantea de esta manera para poder consumir los datos desde diferentes plataformas, y de una manera escalonada. En nuestro caso el api Javascript hará una petición al Servlet, que a su vez consultará los datos mediante el SOAP.

6.3.1 Servicio SOAP

Es un servicio web basado en el protocolo estándar SOAP que define como dos procesos pueden comunicarse mediante datos XML. Para su desarrollo me he apoyado en el framework de código abierto Apache Axis[8], el que a partir de unas clases que tenemos que programar, nos facilita la comunicación de los servicios SOAP.

El servicio, bajo petición, se encarga de conectarse a la BBDD, mediante la herramienta llamada Hibernate[9] de mapeo de base de datos y objetos Java, y devolver el listado de incidencias.

A continuación se presenta el modelo de clases generado, y una breve explicación de las clases que toman parte en este servicio. Para información más detallada sobre el servicio y las herramientas utilizadas dirigirse al Anexo V.

[8] **Apache Axis** es un framework de código abierto, basado en XML para servicios web. Consiste en una implementación en Java servidor SOAP, así como diversos utilitarios y APIs para generar y desplegar aplicaciones de servicios web.

[9] **HIBERNATE** es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java, que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación (Fuente: http://es.wikipedia.org/wiki/Hibernate)

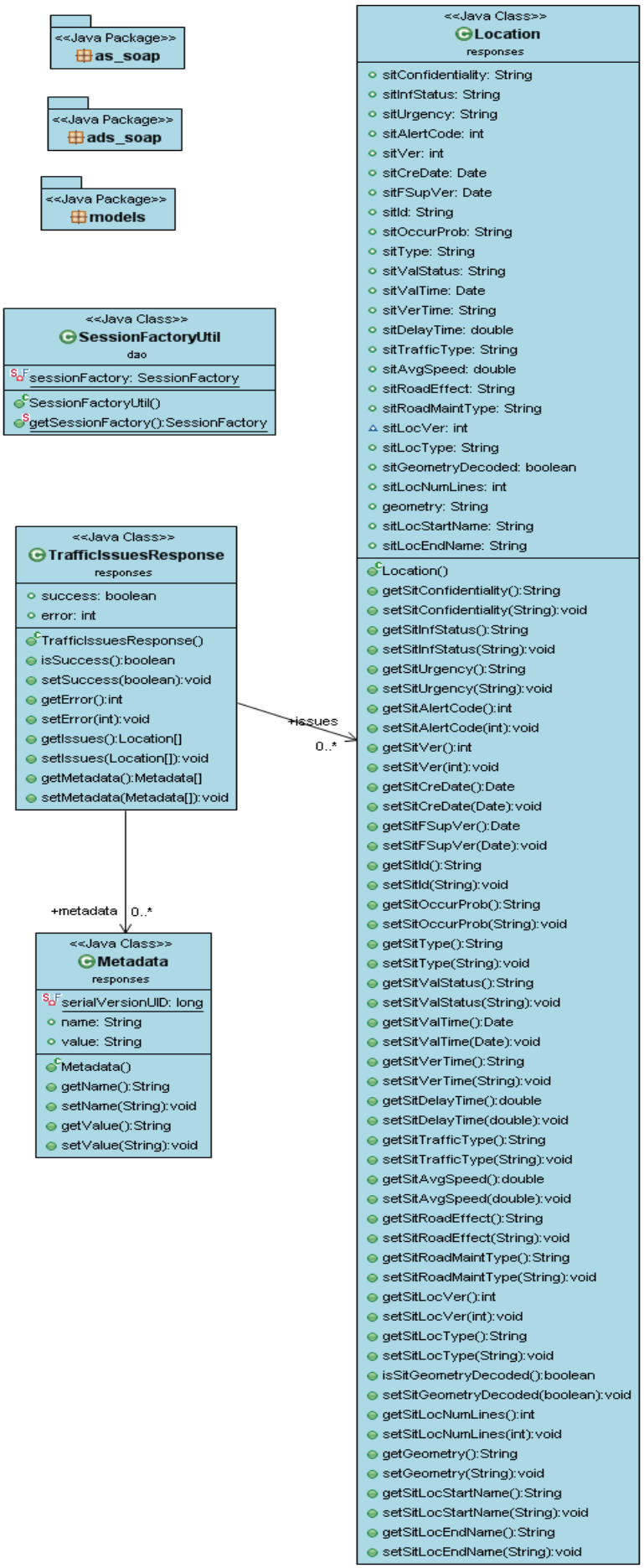

<<Java Package>>
as_soap
<<Java Package>>
ads_soap
<<Java Package>>
models
<<Java Class>>
SessionFactoryUtil
dao
sessionFactory: SessionFactory
SessionFactoryUtil()
getSessionFactory():SessionFactory
<<Java Class>>
TrafficIssuesResponse
responses
success: boolean
error: int
TrafficIssuesResponse()
isSuccess():boolean
setSuccess(boolean):void
getError():int
setError(int):void
getIssues():Location[]
setIssues(Location[]):void
getMetadata():Metadata[]
setMetadata(Metadata[]):void
+issues
0..*
+metadata
0..*
<<Java Class>>
Metadata
responses
serialVersionUID: long
name: String
value: String
Metadata()
getName():String
setName(String):void
getValue():String
setValue(String):void
<<Java Class>>
Location
responses
sitConfidentiality: String
sitInfStatus: String
sitUrgency: String
sitAlertCode: int
sitVer: int
sitCreDate: Date
sitFSupVer: Date
sitId: String
sitOccurProb: String
sitType: String
sitValStatus: String
sitValTime: Date
sitVerTime: String
sitDelayTime: double
sitTrafficType: String
sitAvgSpeed: double
sitRoadEffect: String
sitRoadMaintType: String
sitLocVer: int
sitLocType: String
sitGeometryDecoded: boolean
sitLocNumLines: int
geometry: String
sitLocStartName: String
sitLocEndName: String
Location()
getSitConfidentiality():String
setSitConfidentiality(String):void
getSitInfStatus():String
setSitInfStatus(String):void
getSitUrgency():String
setSitUrgency(String):void
getSitAlertCode():int
setSitAlertCode(int):void
getSitVer():int
setSitVer(int):void
getSitCreDate():Date
setSitCreDate(Date):void
getSitFSupVer():Date
setSitFSupVer(Date):void
getSitId():String
setSitId(String):void
getSitOccurProb():String
setSitOccurProb(String):void
getSitType():String
setSitType(String):void
getSitValStatus():String
setSitValStatus(String):void
getSitValTime():Date
setSitValTime(Date):void
getSitVerTime():String
setSitVerTime(String):void
getSitDelayTime():double
setSitDelayTime(double):void
getSitTrafficType():String
setSitTrafficType(String):void
getSitAvgSpeed():double
setSitAvgSpeed(double):void
getSitRoadEffect():String
setSitRoadEffect(String):void
getSitRoadMaintType():String
setSitRoadMaintType(String):void
getSitLocVer():int
setSitLocVer(int):void
getSitLocType():String
setSitLocType(String):void
isSitGeometryDecoded():boolean
setSitGeometryDecoded(boolean):void
getSitLocNumLines():int
setSitLocNumLines(int):void
getGeometry():String
setGeometry(String):void
getSitLocStartName():String
setSitLocStartName(String):void
getSitLocEndName():String
setSitLocEndName(String):void

Diagrama 25: Diagrama de clases Servicio SOAP

Tts.java

Es la clase principal del servicio. En ella se especifican los métodos que soporta el servicio SOAP, y se definen una serie de métodos de validación de usuario, que para nuestro sistema no tendrán sentido. Esta clase delega en otras la funcionalidad de acceso a datos.

IDao.java <Interface>

Es una interface que define una serie de métodos de acceso a datos.

Dao.java

Esta clase implementa la interface anteriormente mencionada "IDao.java". Es instanciada por la clase general "Tts.java", y se definen los métodos de acceso a datos. Para ello, como ya se ha comentado, utiliza la herramienta Hibernate. Mediante los objetos generados por esta herramienta, accede a la base de datos y genera los objetos de respuesta para devolvérselos a "Tts.java" quien será la encargada de generar la respuesta del servicio.

SessionFactoryUtil.java

Una clase que sigue el patrón Singleton, encargada de obtener la conexión con la base de datos. Esta conexión se hace mediante la herramienta Hibernate, que genera una sesión con la base de datos, y no se cierra hasta caducar. Con eso lo que se consigue es gestionar las conexiones hacia la base de datos mediante un pool de conexiones[10], internamente Hibernate utiliza uno llamado C3P0.

hibernatePostgres.cfg.xml

Fichero xml de configuración de la conexión a BBDD que utiliza Hibernate.

[10] **Pool de conexiones,** es una cache de conexiones a base de datos activas, de tal manera que estas conexiones pueden ser reutilizadas en peticiones futuras a la base de datos

hibernate.reveng.xml

Fichero xml de Hibernate, en el que se indican las tablas de base de datos que se quieren mapear a objetos Java.

Realtimetraffic.hbm.xml y Realtimetraffic.java

Fichero xml generado por Hibernate que se corresponde con una tabla de la base de datos, en este caso la tabla donde se almacenan las incidencias, y la clase Java generada por la herramienta basándose en ese fichero xml.

TrafficIssuesResponse.java

Objeto java de respuesta del servicio SOAP. Este objeto es mapeado a xml por la herramienta Apache Axis para generar la respuesta del servicio. En este objeto se definen una serie de atributos que indican información general sobre la petición, como si todo ha salido correctamente y un código de error en caso contrario, y dos clases java de respuesta, un listado de incidencias "Location.java", en la que se indican los datos de cada incidencia, y otra "Metadata.java", en la que se añade información extendida, como el tiempo de respuesta y la versión del servicio.

6.3.2 Servicio Servlet

Es un servicio basado en los tan conocidos objetos Servlet de java. En nuestro sistema este servicio se encarga de hacer de intermediario entre la aplicación web, la parte cliente, y el servicio SOAP.

De una forma muy general, el servicio se mantiene a la escucha hasta que recibe una petición, la redirige al SOAP, recibe la respuesta, y la reenvía. Para realizar todo esto, como ya hemos comentado antes, nos apoyamos en la herramienta Apache Axis 2, la cual nos crea una serie de objetos que utilizaremos para la transferencia de información en xml entre el servicio Servlet y el SOAP.

Para minimizar el proceso de respuesta en el cliente, la respuesta se hará en formato JSON, utilizando una librería de Java llamada *json-org* que nos facilitará la creación de ese tipo de respuesta.

A continuación se presenta el modelo de clases generado, y una breve explicación de las clases que toman parte en este servicio. Para información más detallada sobre el servicio y las herramientas utilizadas dirigirse al Anexo VI.

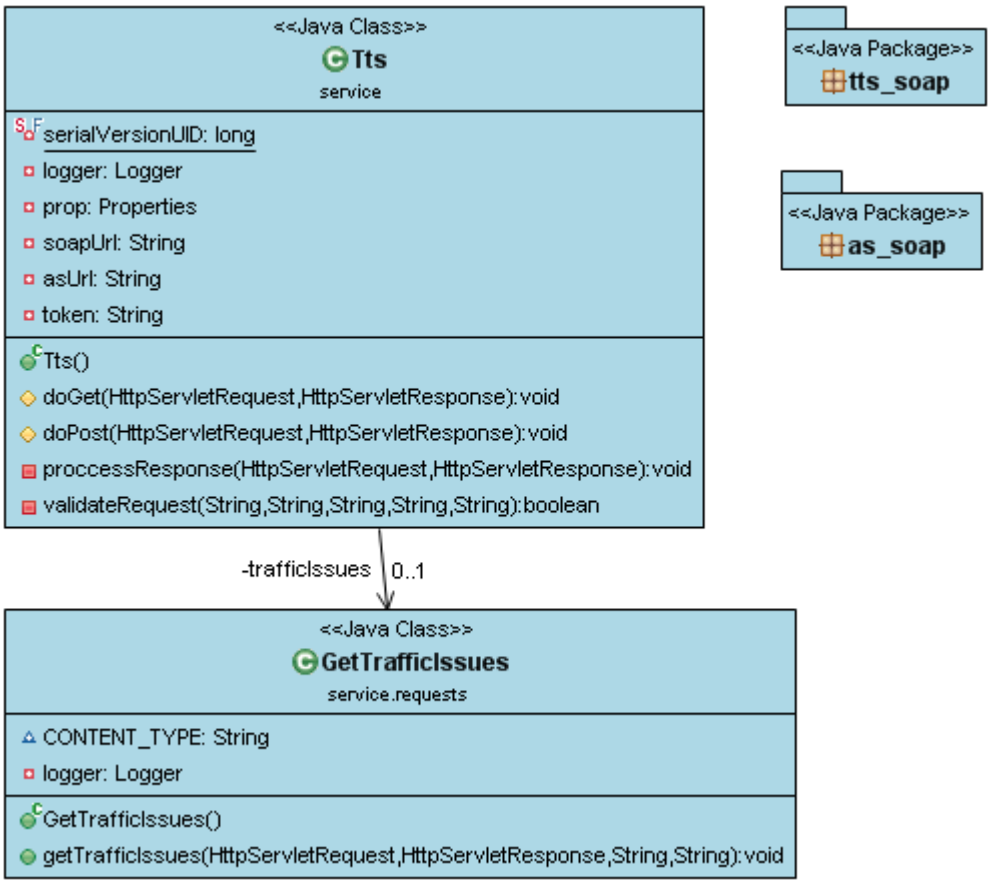


Diagrama 26: Diagrama de clases Servicio Servlet

Tts.java

La clase principal que implementa la interface *HttpServlet*. Se encarga de recibir la petición y de delegar en otra clase la lógica de comunicación con el SOAP y de creación de la respuesta. Por lo que se podría decir que se encarga de gestionar las diferentes funcionalidades que ofrece. Esto se decidió hacer de esta forma por estructurar el servicio, y así, poder hacerlo extensible con nuevas funcionalidades sin que el código de esta clase crezca demasiado.

GetTrafficIssues.java

De momento es la única funcionalidad que ofrece este servicio, pero la idea es que pueda tener otras. La clase principal delegará en esta clase la lógica del servicio de recuperar las incidencias de tráfico obtenidas mediante el servicio web SOAP. Cuando recibe la respuesta, la parsea a formato JSON y la devuelve.

Paquetes tts_soap y as_soap

Estos dos paquetes son autogenerados mediante la librería Apache Axis 2 y son los encargados de gestionar la comunicación entre los servicios web. Para nuestro sistema, el *as_soap* sería transparente, ya que se encarga de la autenticación que en nuestro caso no se evalúa. El otro paquete, *tts_soap,* realizará la conexión con el SOAP definido anteriormente.

6.3.3 API Javascript

Se decidió crear este API para facilitar las comunicaciones entre la aplicación web y el servicio Servlet. Permite hacer peticiones utilizando los métodos POST y GET que ofrecen los diferentes servidores.

Para el método POST utilizamos un mecanismo llamado CORS[11], que permite realizar peticiones utilizando Javascript a servidores alojados en otros dominios, para evitar el CROSS-DOMAIN[12].

Para el método GET, utilizamos otra técnica conocida como JSONP[13], que se trata en enviar en la petición un parámetro de callback y encapsular la respuesta en un objeto JSON que se incrusta en la cabecera del documento HTML, de esta forma al recibir la respuesta se ejecuta la función de callback. Esto se realiza para evitar el CROSS-DOMAIN entre dominios diferentes.

Para información más detallada sobre este API y las técnicas que utiliza, dirigirse al Anexo VII.

[11] **CORS** es un mecanismo que permite que los Javascript de una página web realicen peticiones XMLHttpRequest a otros dominios que no sean donde se encuentra alojada dicha pagina web

[12] **CROSS-DOMAIN** protocolo de seguridad que ofrece la posibilidad de acceder o transferir información entre dos o más dominios de seguridad diferentes, estas peticiones serán denegadas por la política de seguridad del mismo origen.

[13] **JSONP** es una técnica utilizada en Javascript para acceder a información alojada en diferentes dominios.

7. Pruebas

Se antoja necesario en toda metodología de desarrollo de software someter al proyecto a una serie de pruebas con el fin de cerciorarse de que hemos logrado un desarrollo fiel a lo esperado y con el mayor número de errores subsanados; se afirma "el mayor número" ya que siempre se cuenta con la posibilidad más que factible de no encontrar todos los errores presentes.

El origen y la proliferación de errores son inherentes a cualquier proyecto informático. Su naturaleza puede ser de muy diversa índole; desde errores tipográficos hasta respuestas incorrectas tras una compleja y fortuita navegación por las páginas o problemas relacionados con el hardware no previstos al desarrollar el proyecto en otra arquitectura.

En la cadena de valor del desarrollo de un software específico, el proceso de prueba es clave a la hora de detectar errores o fallos. Conceptos como estabilidad, escalabilidad, eficiencia y seguridad se relacionan a la calidad de un producto bien desarrollado. Las aplicaciones de software han crecido en complejidad y tamaño, y por consiguiente también en costos. Hoy en día es crucial verificar y evaluar la calidad de lo construido con el fin de minimizar el costo de su reparación. Mientras antes se detecte un fallo, más barato es su corrección.

En caso de no superar alguna de las pruebas se debe volver a un estado anterior al desarrollo para solucionar el problema con el consiguiente gasto de tiempo y recursos, algo nada deseable pero necesario al fin y al cabo.

A continuación se comentarán los tipos de prueba efectuados. En general se creará un proyecto para realizar pruebas para cada servicio o librería generada. Para la parte cliente, las pruebas se efectuarán navegando por las diferentes opciones que ofrece la aplicación web.

7.1 Librería "Parser"

Para esta librería, se ha generado un proyecto en Eclipse llamado "tts_tester". Este proyecto dispone de una sola clase, llamada *RTTrafficFeedTester.java* que se encarga de realizar todo el proceso de parseo de incidencias y almacenamiento en la base de datos.

Se debe activar el log de la librería parser en modo depuración, y ejecutar la clase de test creada. Con esto, lo que se consigue es un proceso de parseado completo, calcula los tiempos de parseo y decodificación total del feed de TomTom, y en caso de algún error, mediante las excepciones controladas en la librería, podremos verificar esos errores y tratar de corregirlos basándonos en las líneas escritas en el fichero log.

Más información sobre estas pruebas en el Anexo IV.

7.2 Servicio en tres niveles

7.2.1 Servicio SOAP

Para este servicio, se genera otro proyecto en Eclipse. Como en todas las pruebas, lo primero es activar el log del servicio en modo depuración, a continuación se despliega el servicio en nuestro servidor configurado en el Eclipse. Utilizando la herramienta Apache Axis 2 integrada en nuestro eclipse, generamos el modelo de

clases para facilitar la petición al servicio SOAP. A continuación se ejecuta nuestro proyecto, el cual está configurado para solicitar información sobre las incidencias y posteriormente mostrar la respuesta en la consola del Eclipse.

Revisando el log, se pueden encontrar los posibles errores producidos durante el proceso del servicio SOAP, una vez detectados se procede a su corrección.

Suele ser muy común que el log no nos de la suficiente información acerca de por qué ha fallado el proceso. En esos casos, el procedimiento a seguir sería arrancar nuestro servidor configurado en el Eclipse en modo depuración, y utilizando la herramienta que ofrece Eclipse, colocar un punto de interrupción en la línea de código que produce el error (esta línea la podemos encontrar en el log) y depurar el código hasta encontrar el fallo.

7.2.2 Servicio Servlet y API Javascript

Estos dos apartados los unimos en uno, ya que probaremos el correcto funcionamiento del servicio Servlet utilizando el API javascript creado. Para ello, generamos una página web muy simple, que importe el API javascript. Esta página, mediante el API, hará una llamada al servicio servlet, y nos mostrará en pantalla los resultados obtenidos.

Como siempre, lo primero será activar el log en modo depuración del servicio, y lo desplegaremos en el servidor configurado en nuestro Eclipse. Crearemos el documento html de la pagina web junto con un pequeño trozo de código en javascript que realizará la llamada y recuperará la respuesta utilizando el API. En caso de errores, se deberá revisar el log del servicio y en caso de que el fallo no esté en el servicio, deberemos depurar nuestra API.

Para depurar el API javascript, utilizaremos el navegador web Mozilla Firefox, en el cual se puede instalar un plugin llamado Firebug, que nos permite depurar el código javascript.

7.3 Aplicación Web

La mejor manera para testear una aplicación web, es navegando por las diferentes posibilidades que ofrece y de esa manera comprobar su correcto funcionamiento. A este tipo de prueba se le denomina *pruebas de caja negra.*

Estas pruebas, a ser posible, es mejor que las realice el cliente del producto final, o de no poder ser de esa forma, alguien externo al proyecto pero con conocimiento del resultado final deseado. De esta manera, una persona externa puede encontrar fallos que el programador o diseñador pueden pasar por alto, y en caso de realizarlas el cliente, puede encontrar diferencias entre lo deseado y el resultado final.

Las pruebas que se pueden realizar serían las siguientes:

- Comprobar que al inicializar la aplicación se muestran incidencias tanto en el panel de resultados como en el mapa.
- Revisar cada cierto tiempo, 2 minutos entre actualización y actualización, que las incidencias mostradas se van actualizando, tanto en el mapa como en el panel de información.

- Colocar el puntero del ratón sobre una incidencia en el mapa, comprobar que el icono de la incidencia se sobresalta, y ver que en el panel de la derecha se carga su información asociada.
- Colocar el puntero del ratón sobre una incidencia en el listado de incidencias del panel, comprobar que se rellena el panel de información, y ver si la incidencia sobre el mapa se resalta.
- Hacer doble click sobre una incidencia en el listado de incidencias de la derecha, comprobar que en el mapa se ha hecho zoom a la zona del mapa donde se encuentra la incidencia.

En caso de encontrarnos con errores, se utilizará el navegador web Mozilla Firefox y su complemento Firebug, para así poder depurar el código javascript y encontrar la causa del error para poder subsanarlo.

7.4 Conclusiones de las pruebas

Una vez realizadas todas las pruebas descritas en este apartado y de la corrección de los fallos que nos hemos encontrado, podemos concluir que el sistema de gestión de incidencias de tráfico desarrollado, cumple satisfactoriamente los requisitos impuestos al comienzo del proyecto.

Para que esto sea así, aunque realizar las pruebas definidas puede resultar un tanto tedioso, es de suma importancia dedicarle el máximo tiempo y la máxima atención posible, ya que nos ayuda a encontrar y subsanar los errores que hayamos podido cometer.

8. Gestión del proyecto

8.1 Asignación de recursos

Representación final de la estimación realizada al inicio del proyecto frente a la dedicación real sobre las siguientes fases del proyecto. Esta estimación puede comprobarse más detalladamente en el apartado 1.4 Planificación

FASES	Estimación		Real	
	Horas	Fecha entrega	Horas	Fecha entrega
1º. DOP	16 horas	08-11-2013	16 horas	08-11-2013
2º. Captura de requisitos	28 horas	22-11-2013	35 horas	27-11-2013
3º. Análisis	30 horas	06-12-2013	36 horas	12-12-2013
4º. Diseño	16 horas	13-12-2013	18 horas	24-12-2013
5º. Desarrollo y pruebas	150 horas	03-02-2014	185 horas	14-02-2014
TOTAL	248 horas		290 horas	

Tabla 5: Asignación de recursos

Como se puede observar la mayor desviación se produce en el desarrollo del sistema. Esto es debido a la inexperiencia y el desconocimiento de alguna de las tecnologías que se han utilizado, como Apache Axis 2, OpenLR o Hibernate, para las que se han tenido que emplear horas para su aprendizaje.

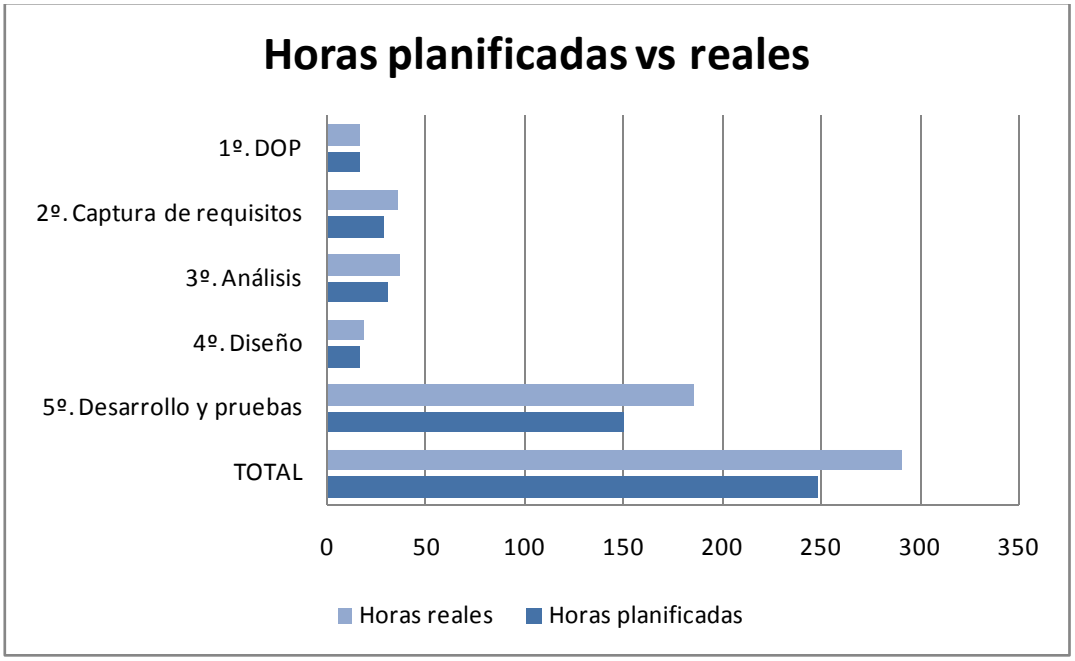


Figura 5: Horas planificadas vs reales

8.2 Gantt planificado vs Gantt real

8.2.1 Planificado

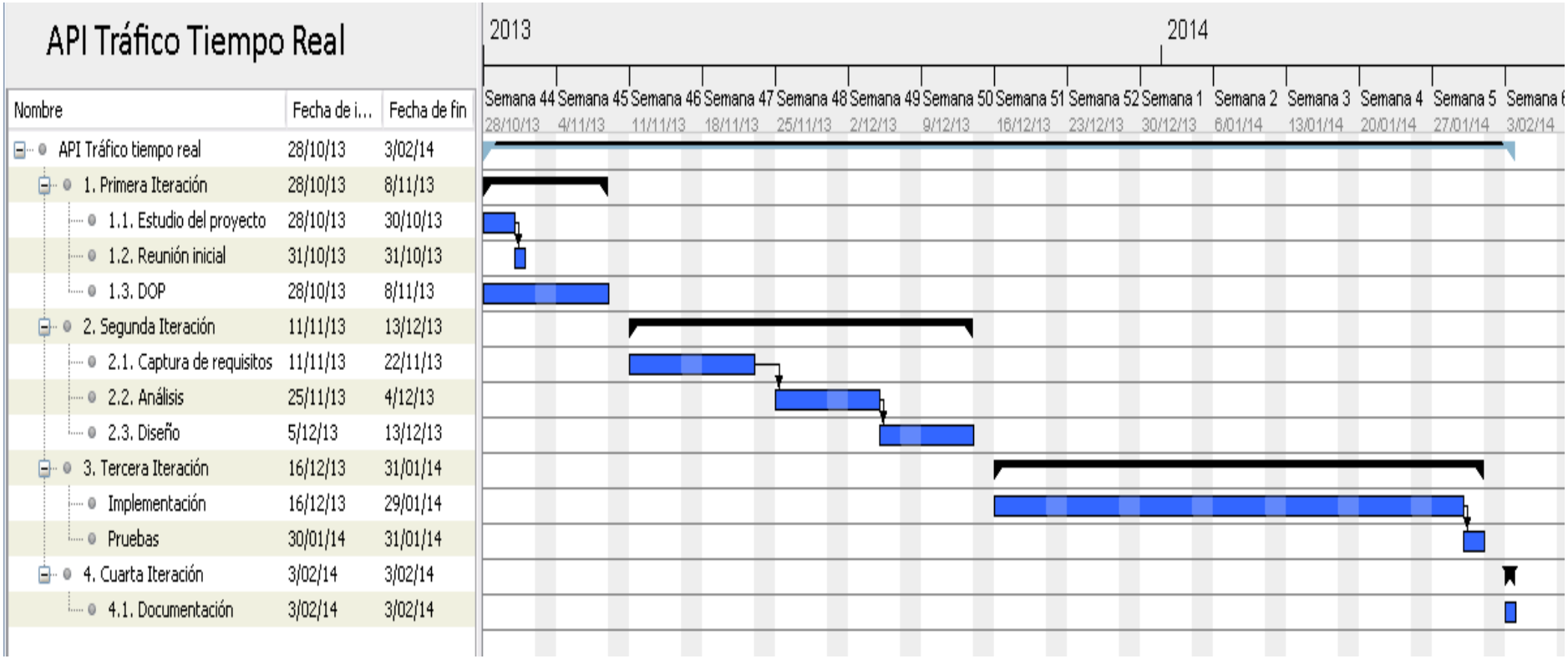


Diagrama 27: Diagrama de Gantt Planificado

8.2.2 Gantt real

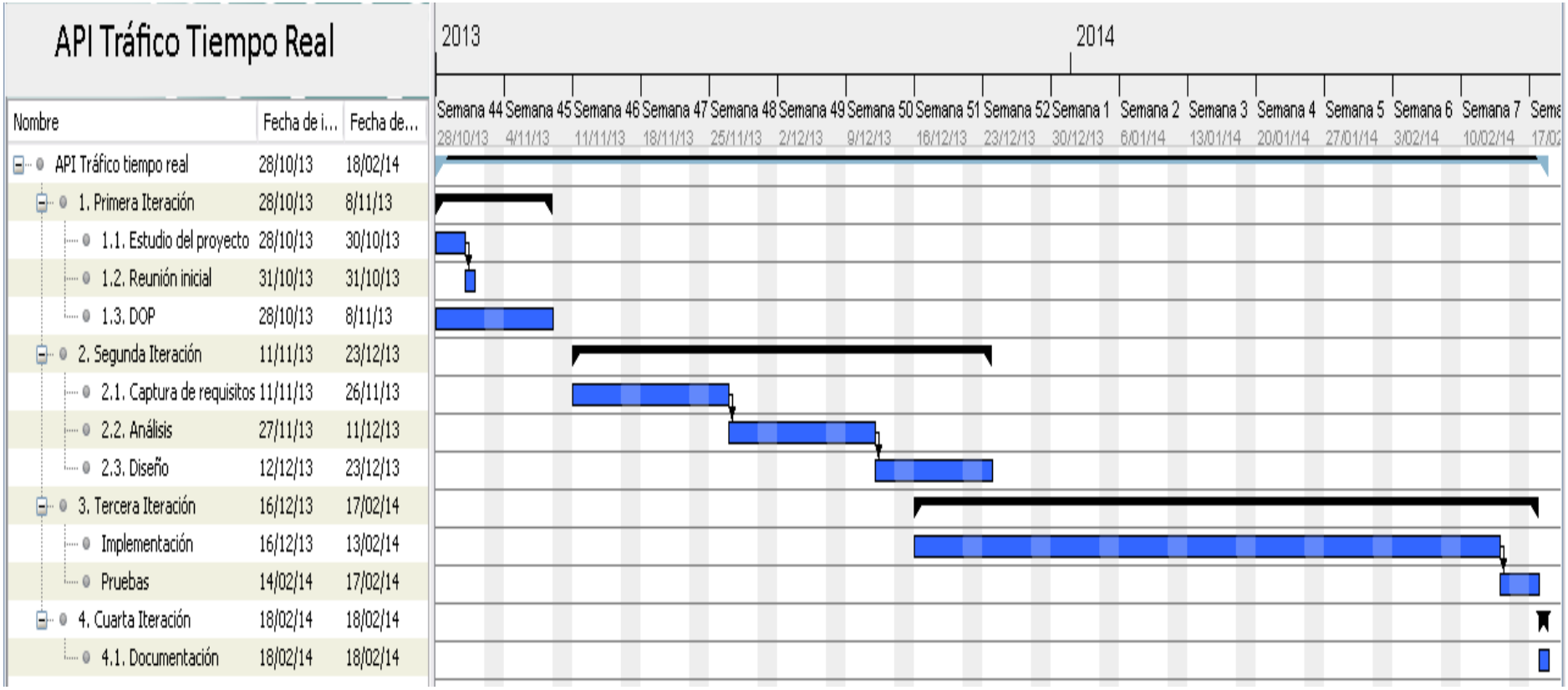


Diagrama 28: Diagrama de Gantt Real

9. Conclusiones

Llegados a este punto ya son conocidos los hitos alcanzados, contenido de las entregas, fechas cumplidas, retrasos arrastrados, etc. Es el momento por tanto de hacer un resumen de los objetivos cumplidos.

9.1 Resumen del proyecto

Al inicio del proyecto se pretendía crear un API que facilitara la integración de información sobre el tráfico en tiempo real de las carreteras del País Vasco en cualquier tipo de sistema, como aplicaciones para teléfonos móviles, aplicaciones web o de escritorio.

Para ello se ha conseguido ofrecer el servicio a tres niveles, servicios web tanto SOAP como Servlet, y un nivel más bajo para aplicaciones web utilizando Javascript, de una forma escalonada y de fácil escalabilidad, es decir, las tres partes aunque estén unidas, son totalmente independientes unas de otras, y realizar mejoras o modificaciones en uno de los niveles, no tendría porque suponer tener que actualizar el resto de niveles.

La mayor dificultad ha residido en la necesidad de desbloquear temas como la decodificación de las incidencias OpenLR, por su poca documentación, y la integración de los servicios web SOAP mediante la herramienta Apache Axis 2.

La parte más sencilla, por así decirlo, ha sido la parte de la aplicación web, gracias a los conocimientos previos y la ayuda del tutor de la empresa. A parte, no se tuvo que realizar una aplicación completa ya que se ha integrado como un módulo sobre el Framework GIS desarrollado previamente por la empresa.

En este punto se podría decir que se han cumplido los objetivos marcados, aunque sin estar dentro de los plazos estimados, debido a las desviaciones que ya se han explicado en el apartado anterior.

9.2 Posibles mejoras

El sistema de incidencias de tráfico en tiempo real tiene muchísimas posibles mejoras, de hecho, al finalizar este proyecto hemos seguido desarrollando más su funcionalidad. Esa funcionalidad no está incluida en este proyecto ya que no estaba planificada al inicio.

Las mejoras realizadas tanto como las que se podrían realizar, van desde la parte cliente, hasta la parte de los servicios y la gestión de las incidencias. Por ejemplo, una mejora posible sería almacenar en la BBDD un histórico de las incidencias decodificadas para su posterior estudio utilizando la herramienta web.

En la parte cliente las mejoras pueden ser infinitas, teniendo en cuenta la interface gráfica y el tema de estilos de los componentes. En su funcionalidad se podría mejorar la interacción con el mapa.

9.3 Valoración personal

En el ámbito personal, la aportación más importante que creo que me llevo de este proyecto es aprender cómo se trabaja en un proyecto real, tan real que la intención de la empresa es vender una mejora del API de tráfico en tiempo real aquí expuesto a los organismos interesados.

Por otro lado, aparte de aprender a utilizar tecnologías y soluciones que se utilizan en el mundo del desarrollo de aplicaciones, como los servicios web SOAP, también se aprende por ejemplo, que las estimaciones son eso, simples estimaciones, aunque siempre intentando ajustarlas al máximo, y que tienen mucha más importancia que la que al principio les damos, al igual que el análisis y el diseño.

Otro tema muy importante son las pruebas. Personalmente al empezar el proyecto no le di apenas importancia a las pruebas finales, pero una vez realizado el desarrollo uno se da cuenta de la importancia que tienen, ya que son vitales para encontrar y solucionar los posibles bugs del sistema.

Por último añadir que me ha gustado mucho trabajar junto a trabajadores que pese a su juventud tienen más de 10 años de experiencia. Creo que he aprendido mucho de ellos, ya que me han aportado mucha ayuda en los momentos en los que me quedaba atascado, y me han surtido de ideas muy interesantes.

10. Anexos

Anexo I: Log de la aplicación

Log4java es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o "logs" (data logging) a tiempo de ejecución y no a tiempo de compilación como es comúnmente realizado.

Para nuestro proyecto, utilizaremos esta librería para hacer un seguimiento del comportamiento de los diferentes procesos que se ejecutan, tanto para los servicios como para las librerías de la parte del servidor, es decir, la capa lógica de negocio.

Para utilizar esta librería, en primer lugar debemos descargarnos el JAR accesible desde su página web, http://logging.apache.org/. Incluimos esta librería en la carpeta de librerías de cada proyecto que vayamos a realizar.

A continuación debemos incluir un fichero *properties* en la raíz del proyecto llamado *loj4java.properties*, en nuestro caso el documento será como el que se incluye a continuación:

```
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=${catalina.base}/logs/tts_soap.log
log4j.appender.file.append=true
log4j.appender.file.DatePattern='.'yyyy-MM-dd
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d (%5p) %-5l - <%m>%n
log4j.rootLogger=debug, file


#httpclient logging proeprties
log4j.logger.httpclient.wire=ERROR

#Hibernate logging proeprties
log4j.logger.org.hibernate=ERROR
log4j.logger.org.hibernate.sql=ERROR
log4j.logger.org.hibernate.hql=ERROR
log4j.category.org.hibernate.dialect=ERROR

#Axis2 logging properties
log4j.logger.org.apache.axis2.enterprise=ERROR
log4j.logger.org.apache.axis2=ERROR
log4j.logger.org.apache.axiom=ERROR
log4j.logger.de.hunsicker.jalopy.io=ERROR
log4j.logger.org.apache.commons.httpclient=ERROR
```

Donde los aspectos más importantes serian:

- `log4j.appender.file.File`: indica el fichero de escritura del log

- `log4j.appender.file.layout.ConversionPattern`: el formato de las líneas escritas.

- `log4j.rootLogger`: el modo en el que queramos que funcione y la plantilla que utilizará. Para el desarrollo se recomienda utilizar el modo *debug* para que nos muestre todos los mensajes introducidos. Una vez se ponga el proyecto en

producción, el modo que se utiliza es *error* para que solo nos muestre los mensajes de error y no tener un fichero de log muy grande.

- Se indican los niveles de log de las librerías externas utilizadas por la aplicación para que en el fichero solo nos escriba lo que nos interesa.

Fragmento de código en el que se puede observar como se utiliza:

```
public class Dao implements IDao{

    //Se declara una variable global para inicializar el logger
    private Logger logger = Logger.getLogger(Dao.class);

    @Override
      public Location[] getTrafficIssues(String types, String bbox)  {
      try{
        logger.info("Inicio método”);
        ArrayList<Location> locations = getLocations(types, bbox);
        if(locations == null){
            logger.info("No se tienen incidencias");
            return null;
        }
        logger.info("Numero de incidencias obtenidas:"+
locations.size());
        Location[] res = new Location[locations.size()];
        int i = 0;
        for(Iterator<Location> iter = locations.iterator();
iter.hasNext();){
            res[i] = iter.next();
            i++;
        }
        logger.info("Fin del método, devolvemos las incidencias");

        return res;
      }catch(Exception oo){
            logger.error("Error: "+oo.getMessage());
            throw oo;
      }
  }
…
```

Salida que produce esta configuración:

```
2013-11-26 16:18:22,653 (DEBUG) dao.Dao.getTrafficIssues(Dao.java:120) -
<Inicio método>

2013-11-26 16:18:22,654 (DEBUG) dao.Dao.getTrafficIssues (Dao.java:126) -
<Numero de incidencias obtenidas: 34>

2013-11-26 16:18:22,659 (DEBUG) service.Dao.getTrafficIssues (Dao.java:132) -
<Fin del método, devolvemos las incidencias>
```

Anexo II: Servicio RealTimeTraffic de TomTom

A continuación me parece interesante presentar el servicio que ofrece TomTom mediante la documentación ofrecida por ellos para poder entender el objetivo del proyecto y su magnitud para posibles utilidades.

Presentación del servicio

Esta documentación es la original distribuida por TomTom, por lo que esta en Ingles. Se presenta aquí como un complemento al proyecto:

Enterprise Traffic

Navigate with up-to-the-minute information

The number of vehicles on the road network is increasing, congestion is building and travel times are lengthening. Drivers are being challenged more frequently, and users of mapping and navigation applications are seeking ways to save time getting to their destinations.

By incorporating TomTom Enterprise Traffic into a navigation solution, drivers can determine the quickest route to their destinations by considering "live" road conditions. The data in each Enterprise Traffic file includes road delays allowing routing programs to evaluate the true travel time to each destination. All relevant congestion information can be displayed on the device screen.

Enterprise Traffic ensures market leading quality by merging multiple data sources. These include anonymous measurements from GPS navigation devices and mobile phone signals, sensor data from governments and journalistic data about incidents. Using a proprietary data fusion method, TomTom efficiently processes these large data sets and instantly transmits the results to industry partners.

Why TomTom Enterprise Traffic?

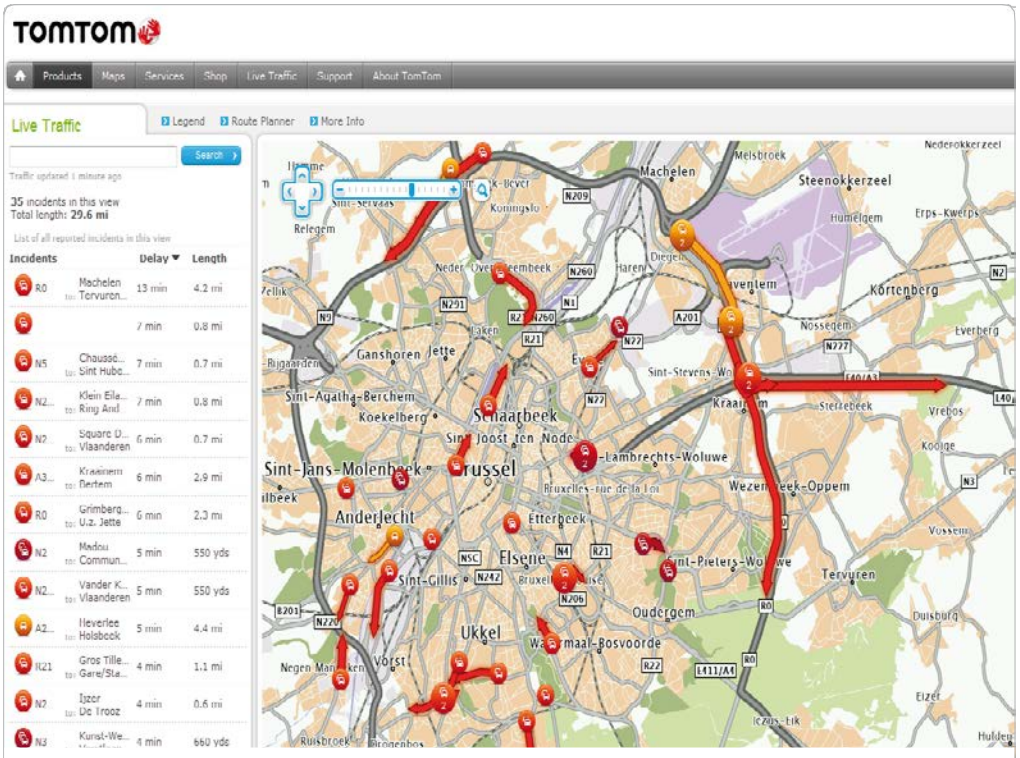
- **Highest jam accuracy**
  merged data from multiple sources
- **Highest road mileage**
  secondary roads in addition to motorways and major roads
- **Fastest refresh rate**
  new file available every minute
- **Favours car drivers**
  predominantly car-centric data sources not distorted by trucks

* Data feed for some countries are currently only available in OpenLR location referencing format. Please ask if interested in TMC availability.

maps & content | real time & historical traffic | lbs

TOMTOM

www.tomtom.com/licensing



Real time traffic can be seen on TomTom Live Traffic (www.tomtom.com/livetraffic)

FEaTurE	BEnEFiT
TMC location referencing	• Ease of integration into existing maps and routing engines, with congestion and incident locations described precisely using local TMC tables and offsets
OpenLr location referencing	• Information is decoded onto any map so congestion and incident locations are described precisely on any road. This includes secondary roads
Datex 2 file format	• Widely accepted industry standard ensures easy integration with existing maps and routing engines
Multiple contributory data sources	• Confidence that data is reliable and accurate
available in multiple countries	• Single source of data for key markets with consistent quality
Platform maturity	• Confidence that data is reliable and accurate – the same data used by 'HD Traffic' on TomTom PNDs since 2007
Fast file update every minute	• Confidence that data is fresh, reliable and accurate

Comprehensive and accurate traffic data adds to the value of a navigation system. This transforms it into an application for use even on regular commutes as it tells drivers of any unusual incidents on their journey, saving time and reducing stress.

hts reserved.

TomTom Enterprise Traffic: Interface description

1 External interface

The Enterprise traffic feed external interfaces are based on DATEX 2. DATEX 2 provides standards for

- Data exchange methods
- Payload definition

Simple HTTP server profile

The interface supports the client pull method, also known as the 'Simple HTTP server' profile. The profile is described in the DATEX 2 DevGuide and ExchangePSM. For download details: see the table above. An exception is the SOAP envelope, described in section 4.4 of the DevGuide, which is not supported. It would be required for interoperability with clients using web services. If a customer requires it, it will be made available on a different URL.

URL

The URL for the different feed types and products is constructed in the following way:

https://SERVERNAME/tsq/PRODUCTNAME/APIKEY/content.xml where:

component	description
SERVERNAME	the name of the server
PRODUCTNAME	the name of the product
APIKEY	a client specific key, identifying the client, used for product authorization

The retrieved content is a DATEX 2 PayloadPublication

Feed type

Different types of content can be identified. For instance, DATEX 2 defines a situation publication for traffic and travel information, a measured data publication for data from

sensors, etc. The feed type determines the format and the content of the output of the feed.

The following feed types are supported:

feed type	description
	Enterprise Traffic Feed
et	contains incident information, both LOS (Level of Service) and non-LOS (other) messages

The content of the feed is described below

Feed content

This chapter describes the content of the traffic feed. The feed contains information about traffic jams, their cause and impact. About accidents and incidents, with an expected end time, if available. And about travel weather warnings. Also there can be traffic management information, like diversion advices, blocked roads.
The information can be split in

- LOS = Level Of Service
- non-LOS = anything else.

2.1 DATEX2 payload, references

The payload in the output is formatted as DATEX2, which is a European standard for the exchange of Traffic and Travel Information. More information on DATEX2 is available from the official datex 2 website http://datex2.eu/ For the payload specification, data dictionary and user guide the following information can be downloaded from the datex 2 website:

URL	description

http://www.datex2.eu/files/DATEXIIv1.0-UserGuide_v1.0.pdf

The DATEX2 User Guide version 1.0

http://www.datex2.eu/files/DATEXIISchema_1_0_1_0.zip
The standard DATEX2 XML schema version 1.0

http://www.datex2.eu/files/DATEXIIv1.0- DataDictionnary_v1.0.zip

A browsable DATEX2 version

1.0 data dictionary

Payload specification

The feed is based on DATEX2 version 1.0. The data model has been extended with some additional parameters. For details on these extensions, see the section on XML schema, extensions. For understanding the different elements and parameters, refer to the DATEX2 data dictionary and the XML schema. The Situation publication is described in more detail in section 4.10 of the Datex2 User guide version 1.0. (For downloading the User Guide: see the link in the table above.). The reader is advised to read this section 4.10 in the User Guide!

Situation publication

The traffic information is provided as a SituationPublication. At any point in time a SituationPublication contains a snapshot of the latest traffic information. No delta mechanism is supported. A situation publication can contain several different situations.

Situation

A situation represents a traffic/travel situation comprising one or more traffic/travel circumstances which are linked by one or more causal relationships and which apply to related locations. Each traffic/travel circumstance is represented by a Situation Record.

Situation Record

A situation record is one element of a situation. It is characterized by values at a given time, defining one version of this element. When these values change, a new version is created. One situation record can be:

- An road or traffic related event (traffic element),
- An operator action,
- A non road event

information, and can
contain:

- Advice,
- Impact details.

A SituationPublication contains zero or more Situation

Unique identifier

Each situation has a unique identifier. Also each situation record has a unique identifier. This identifier is established when the situation or situation record is first created in the DATEX II's system database. It keeps this identifier within that system for all its life. The unique identifier has a fixed prefix "TT", followed by a string. No further assumptions can be made on the contents of the identifier. The identifier is unique across all supported countries.

Validity period

A situation record has a validity period composed of

- a mandatory overallStartTime, which indicates the start of the situation record, if it is in the past, or an expected start, if it is in the future.
- an optional overallEndTime, if a there is an expected end time.

Impact

If the current delay is available for a situation record, the delayTimeValue will be provided in the impact. If there is information about closed lanes, because of an accident or other incident, that information can be filled in in the impact.

Related situation records

In some cases there are multiple situation records that are related to each other. For instance, there can be a traffic jam caused by an accident. If the locations of these situation records overlap or are connected AND they have the same direction, the situation records will be put in the same situation. In that case a single situation will contain more than one situation record.

Also it is possible that two situation records are related to each other but do not overlap. For instance, one situation record contains a diversion advice, located at a highway
intersection, because further down the road the road is blocked, which is described in a different situation record. In that case the situationPublication will not contain a reference from one situation record to the other.

Language specific texts

A situation record may contain language specific texts. This can for instance be a diversion advice, provided by the network manager, or some other free text.
Typically this text will only be available in the language of the country.

Location

Every Situation Record contains a Group of Locations, which describes the involved location. The Location is described in section 4.16 of the DATEX 2 User Guide. The reader is advised to read this section 4.16 in the User Guide.

Alert-C

The location referencing method used in the feed is Alert-C. See section 4.16.5 in the User Guide. An Alert-C location reference contains a reference to a specific version of an Alert-C Location Table. Alert-C allows referring to an AREA, POINT and LINEAR location. Either Method2 or Method4 is used in the feed, depending on the availability of offset(s). If offset information is available Method4 is used to identify an exact location. If the exact length of the location is available, it is provided as lengthAffected in a supplementaryPositionalDescription

The feed is based on DATEX2 version 1.0. The XML schema is called DATEXIISchema_1_0_1_0_HDT.xsd It is an extension of the standard XML schema: DATEXIISchema_1_0_1_0.xsd. It is a "Level B" extension, which means that it is interoperable with the "Level A" data model (see section 2.2.3 in the User Guide).

Extensions

The DATEX2 data model has been extended to allow additional parameters. The feed has the following extensions:

parameter	description	part of
averageSpeed	the average speed along the location, in km/h	AbnormalTrafficExtensionType
alertCEventCode	the alert-C event code describing the situationRecord. There can be zero or more codes.	SituationRecordExtensionType

The alert-C event code is a number from 1 to 2047. Its meaning is defined in ISO14819 part 2. It is used in RDS-TMC transmission.
If for an AbnormalTraffic situation record the delayTimeValue, the lengthAffected and the averageSpeed are provided, more parameters can be derived:

- travelTime can be derived from lengthAffected and averageSpeed
- freeFlowTravelTime can be derived from travelTime and delay
- freeFlowSpeed can be derived from freeFlowTravelTime and lengthAffected

Example product

A product is a feed of information. It has:

- a PRODUCTNAME
- a feed type, determining the format and contents of the output
- a covered area
- a set of road classes in the output
- an update frequency, that says how often the output is refreshed The table below contains some examples of products.

PRODUCTNAME	feed type	covered area	road class	update frequency
et/nl	Enteprise Traffic	Netherlands	all TMC roads	1 minute
et/de	Enteprise Traffic	Germany	all TMC roads	1 minute

A set of standard products will be specified and maintained. Furthermore, customer specific products can be defined together with a customer.

Every customer recieves its customer specific API key. The API key provides access

to one or more products for a specific time period. The API key needs to be filled in in the URL.

Documento de incidencias

Fragmento del documento xml obtenido a partir del servicio de tráfico en tiempo real ofrecido por TomTom, del cual se obtiene el listado de incidencias. En este fragmento podemos ver los parámetros generales del documento y dos incidencias completas marcadas con la etiqueta <situation>:

```
?xml version="1.0"?>
<d2LogicalModel xmlns="http://datex2.eu/schema/1_0/1_0"
modelBaseVersion="1.0">
  <exchange>
    <supplierIdentification>
      <country>nl</country>
      <nationalIdentifier>TomTom Tele Atlas HD Traffic
Service</nationalIdentifier>
    </supplierIdentification>
  </exchange>
  <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="SituationPublication" lang="en">
    <publicationTime>2012-01-30T17:46:26+01:00</publicationTime>
    <publicationCreator>
      <country>other</country>
      <nationalIdentifier>8ac364ea-fdc5-402e-8580-
6e7a9222455d</nationalIdentifier>
    </publicationCreator>
    <situation id="TTI-8ac364ea-fdc5-402e-8580-6e7a9222455d-
TTL3176130">
      <headerInformation>
        <confidentiality>internalUse</confidentiality>
        <informationStatus>real</informationStatus>
        <urgency>urgent</urgency>
      </headerInformation>
      <situationRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="AbnormalTraffic" id="TTI-8ac364ea-fdc5-402e-8580-
6e7a9222455d-TTL3176130-1">
        <situationRecordCreationTime>2012-01-
30T16:45:26Z</situationRecordCreationTime>
        <situationRecordVersion>1</situationRecordVersion>
        <situationRecordVersionTime>2012-01-
30T17:46:26+01:00</situationRecordVersionTime>
        <situationRecordFirstSupplierVersionTime>2012-01-
30T17:46:26+01:00</situationRecordFirstSupplierVersionTime>
        <probabilityOfOccurrence>probable</probabilityOfOccurrence>
        <validity>
          <validityStatus>active</validityStatus>
          <validityTimeSpecification>
            <overallStartTime>2012-01-30T16:45:26Z</overallStartTime>
          </validityTimeSpecification>
        </validity>
        <impact>
          <delays>
            <delayTimeValue>98.0</delayTimeValue>
          </delays>
        </impact>
        <generalPublicComment>
          <comment>
            <value lang="en-UK">stationary traffic</value>
```

```
          </comment>
        </generalPublicComment>
        <groupOfLocations>
          <locationContainedInGroup xsi:type="Linear">
            <locationExtension>
              <openlr>
                <binary version="3">CwF9tR1mNx55B/4WAKkeS0s=</binary>
              </openlr>
            </locationExtension>
          </locationContainedInGroup>
        </groupOfLocations>
        <situationRecordExtension>
          <alertCEventCode>101</alertCEventCode>
        </situationRecordExtension>
        <abnormalTrafficType>stationaryTraffic</abnormalTrafficType>
        <abnormalTrafficExtension>
          <averageSpeed>10.0</averageSpeed>
        </abnormalTrafficExtension>
      </situationRecord>
    </situation>
    <situation id="TTI-8ac364ea-fdc5-402e-8580-6e7a9222455d-
TTL3176032">
      <headerInformation>
        <confidentiality>internalUse</confidentiality>
        <informationStatus>real</informationStatus>
        <urgency>urgent</urgency>
      </headerInformation>
      <situationRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="AbnormalTraffic" id="TTI-8ac364ea-fdc5-402e-8580-
6e7a9222455d-TTL3176032-1">
        <situationRecordCreationTime>2012-01-
30T16:45:26Z</situationRecordCreationTime>
        <situationRecordVersion>1</situationRecordVersion>
        <situationRecordVersionTime>2012-01-
30T17:46:26+01:00</situationRecordVersionTime>
        <situationRecordFirstSupplierVersionTime>2012-01-
30T17:46:26+01:00</situationRecordFirstSupplierVersionTime>
        <probabilityOfOccurrence>probable</probabilityOfOccurrence>
        <validity>
          <validityStatus>active</validityStatus>
          <validityTimeSpecification>
            <overallStartTime>2012-01-30T16:45:26Z</overallStartTime>
          </validityTimeSpecification>
        </validity>
        <impact>
          <delays>
            <delayTimeValue>146.0</delayTimeValue>
          </delays>
        </impact>
        <generalPublicComment>
          <comment>
            <value lang="en-UK">queuing traffic (with average speeds
Q)</value>
          </comment>
        </generalPublicComment>
        <groupOfLocations>
          <locationContainedInGroup xsi:type="Linear">
            <locationExtension>
              <openlr>
                <binary version="3">C/wmVhqKoRtjBwFsAOAbdVAE</binary>
              </openlr>
```

```
            </locationExtension>
          </locationContainedInGroup>
        </groupOfLocations>
        <situationRecordExtension>
          <alertCEventCode>108</alertCEventCode>
        </situationRecordExtension>
        <abnormalTrafficType>queueingTraffic</abnormalTrafficType>
        <abnormalTrafficExtension>
          <averageSpeed>5.0</averageSpeed>
        </abnormalTrafficExtension>
      </situationRecord>
    </situation>
...
```

Anexo III: Libreria “Decoder"

Documento “Database.xml”

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <!-- Datos de conexión-->
    <entry key="port">5432</entry>
    <entry key="host">localhost</entry>
    <entry key="password">hen.7822</entry>
    <entry key="user">postgres</entry>

    <!-- Campos de "jc" -->
    <entry key="Node.Name">jc</entry>
    <entry key="Node.Column.Id">id</entry>
    <entry key="Node.Column.Longitude">lon</entry>
    <entry key="Node.Column.Latitude">lat</entry>
    <entry key="Node.Column.Geometry">the_geom</entry>

    <!-- Campos de "nw" -->
    <entry key="Line.Name">nw</entry>
    <entry key="Line.Column.Id">id</entry>
    <entry key="Line.Column.StartNodeId">f_jnctid</entry>
    <entry key="Line.Column.EndNodeId">t_jnctid</entry>
    <entry key="Line.Column.DisplayName">name</entry>
    <entry key="Line.Column.DisplayNameLanguageCode">namelc</entry>
    <entry key="Line.Column.FormOfWay">fow</entry>
    <entry key="Line.Column.FunctionalRoadClass">frc</entry>
    <entry key="Line.Column.LengthMeters">meters</entry>
    <entry key="Line.Column.Geometry">the_geom</entry>
    <entry key="Line.Column.OneWay">oneway</entry>

    <!-- Consulta que devulve el numero de nodos totales de "jc" -->
    <entry key="Node.Count">
        SELECT
            Count(*)
        FROM
            {Node.Name}
    </entry>
    <!-- Consulta que dado un identificador de un nodo de "jc"
devuelve su geometria como x,y -->
    <entry key="Node.Statement.Select">
        SELECT
            x({Node.Column.Geometry}) as {Node.Column.Longitude},
```

```
            y({Node.Column.Geometry}) as {Node.Column.Latitude}
        FROM
            {Node.Name}
        WHERE
            {Node.Column.Id} = ?
    </entry>
    <!-- Consulta que devuelve todos los identificadore de los nodos
de "jc" -->
    <entry key="Node.Statement.Select.All">
        SELECT
            {Node.Column.Id}
        FROM
            {Node.Name}
    </entry>
    <!-- Consulta que devuelve la extensión total de todos los nodos
de "jc" -->
    <entry key="Node.Statement.BoundingBox">
        SELECT
            SELECT
                xmin(st_envelope({Node.Column.Geometry})) as xmin,
                xmax(st_envelope({Node.Column.Geometry})) as xmax,
                ymin(st_envelope({Node.Column.Geometry})) as ymin,
                ymax(st_envelope({Node.Column.Geometry})) as ymax,
        FROM
            {Node.Name}
    </entry>
    <!-- Consulta que devuelve un listado de nodos de "jc" contenidos
en una extensión dada -->
    <entry key="Node.Statement.FindCloseBy">
        SELECT
            {Node.Column.Id}
        FROM
            {Node.Name}
        WHERE
            x({Node.Column.Geometry}) &gt;= ? AND
            y({Node.Column.Geometry}) &gt;= ? AND
            x({Node.Column.Geometry}) &lt;= ? AND
            y({Node.Column.Geometry}) &lt;= ?
    </entry>
    <!-- Consulta que devuelve el numero total de tramos de "nw" -->
    <entry key="Line.Count">
        SELECT
            Count(*)
        FROM
            {Line.Name}
    </entry>

    <!-- Consulta que dado un identificador de un tramo de "nw"
devuelve su información asociada-->
    <entry key="Line.Statement.Select">
        SELECT
            {Line.Column.StartNodeId},
            {Line.Column.EndNodeId},
            {Line.Column.FormOfWay},
            {Line.Column.FunctionalRoadClass},
            {Line.Column.LengthMeters},
            {Line.Column.DisplayName},
            {Line.Column.DisplayNameLanguageCode}
            st_asewkb(st_geometryn({Line.Column.Geometry},1)) as
{Line.Column.Geometry}
        FROM
```

```
            {Line.Name}
        WHERE
            {Line.Column.Id} = ?
    </entry>

    <!-- Consulta que dado un identificador de un nodo de "jc",
devuelve el listado de tramos de "nw"
    que tienen dicho nodo como vertice de fin -->
    <entry key="Line.Statement.Incoming">
        SELECT
            {Line.Column.Id},
            {Line.Column.StartNodeId},
            {Line.Column.EndNodeId},
            {Line.Column.FormOfWay},
            {Line.Column.FunctionalRoadClass},
            {Line.Column.LengthMeters},
            {Line.Column.DisplayName},
            {Line.Column.DisplayNameLanguageCode}
            st_asewkb(st_geometryn({Line.Column.Geometry},1)) as
{Line.Column.Geometry}
        FROM
            {Line.Name}
        WHERE
            {Line.Column.EndNodeId} = ?
    </entry>

    <!-- Consulta que dado un identificador de un nodo de "jc",
devuelve el listado de tramos de "nw"
    que tienen dicho nodo como vertice de inicio -->
    <entry key="Line.Statement.Outgoing">
        SELECT
            {Line.Column.Id},
            {Line.Column.StartNodeId},
            {Line.Column.EndNodeId},
            {Line.Column.FormOfWay},
            {Line.Column.FunctionalRoadClass},
            {Line.Column.LengthMeters},
            {Line.Column.DisplayName},
            {Line.Column.DisplayNameLanguageCode}
        FROM
            {Line.Name}
        WHERE
            {Line.Column.StartNodeId} = ?
    </entry>
    <!-- Consulta que dado un identificador de una tramo de "nw"
devuelve la geometria en
    formato de texto asociada a dicho tramo -->
    <entry key="Line.Statement.Geometry">
        SELECT
            st_asewkb(st_geometryn({Line.Column.Geometry},1)) as
{Line.Column.Geometry}
        FROM
            {Line.Name}
        WHERE
            {Line.Column.Id} = ?
    </entry>
    <!-- Consulta que devuelve el listado total de identificadores de
los tramos de "nw" -->
    <entry key="Line.Statement.Select.All">
        SELECT
            {Line.Column.Id}
```

```
        FROM
            {Line.Name}
    </entry>
    <!-- Consulta que devuelve el listado de tramos de "nw" contenidos
dentro de una extensión dada -->
    <entry key="Line.Statement.FindCloseBy">
        SELECT
            {Line.Column.Id}
        FROM
            {Line.Name}
        WHERE
            xmax({Line.Column.Geometry}) &gt;= ? AND
            ymax({Line.Column.Geometry}) &gt;= ? AND
            xmin({Line.Column.Geometry}) &lt;= ? AND
            ymin({Line.Column.Geometry}) &lt;= ?
    </entry>
</properties>
```

Utilización de la configuración "Database.xml"

Código para obtener la extensión total del mapa. En este trozo de código se puede observar como recuperar una consulta definida en el fichero Database.xml y ejecutarla.

```
//Obtiene la extension total de la BD
    @Override
    public Rectangle2D.Double getMapBoundingBox() {
         PreparedStatement ps = null;
            ResultSet rs = null;
            Rectangle2D.Double rect = new Rectangle2D.Double();
            try {
              ps = this.connection.prepareStatement(ConfigurationBBDD.
                    get("Node.Statement.BoundingBox"));

              rs = ps.executeQuery();

              while (rs.next()) {
                double minLon = rs.getDouble("xmin");
                double minLat = rs.getDouble("ymin");
                double maxLon = rs.getDouble("xmax");
                double maxLat = rs.getDouble("ymax");
                rect.setRect(minLon, minLat, maxLon - minLon,
                     maxLat - minLat);
              }
            } catch (SQLException e) {
              e.printStackTrace();
              closeQuietly(rs, ps);
            } finally {
              closeQuietly(rs, ps);
            }
            return rect;
    }
```

Conexión a la BBDD

Código de conexión a la BBDD implementado en la clase MapDatabaseImpl.java:

```
    this.host = ConfigurationBBDD.get("host");
    this.port = ConfigurationBBDD.get("port");
```

```
    this.user = ConfigurationBBDD.get("user");
    this.pass = ConfigurationBBDD.get("password");
    this.bbdd = db;
    try
    {
      String url = "jdbc:postgresql://" + this.host + ":" + this.port
          + "/" + this.bbdd;
      this.conn = DriverManager.getConnection(url, this.user,
          this.pass);
      this.conn.setReadOnly(true);
    } catch (ClassNotFoundException e) {
      throw new IllegalStateException("Unable to load database
          driver.", e);
    } catch (SQLException e) {
      throw new IllegalStateException("Unable to open database
          connection.", e);
    }
```

Cacheado de nodos y lineas

Los siguientes dos métodos se encargan de almacenar en memoria una serie de líneas o nodos, en modo cache, llamada *LRUCache*, para así liberar a la base de datos de transacciones, ya que la librería OpenLR realiza muchas consultas en la base de datos para poder decodificar una incidencia.

```
//Añade una linea a la cache
    private void cacheLine(Line line)
      {
        if (line == null) {
          throw new IllegalArgumentException("Unable to cache null
object.");
        }

        if (this.cachedLines.size() >= 2000)

this.cachedLines.remove(this.cachedLines.keySet().iterator().next());

        this.cachedLines.put(Long.valueOf(line.getID()), line);
      }
    //Añade un nodo a la cache
      private void cacheNode(Node node)
      {
        if (node == null) {
          throw new IllegalArgumentException("Unable to cache null
object.");
        }

        if (this.cachedNodes.size() >= 2000)

this.cachedNodes.remove(this.cachedNodes.keySet().iterator().next());

        this.cachedNodes.put(Long.valueOf(node.getID()), node);
      }
```

Decodificación de una incidencia

Código utilizado para la llamada de decodificación implementado en la clase OpenLRBinaryDecoder.java.

```
public Location decode(String binaryCode){
        this.mdb.start = true;

        LocationReference locRef = null;
        try {
                ByteArray ba = new ByteArray(binaryCode);
                locRef = new LocationReferenceBinaryImpl("LocationID",
                        ba);
        } catch (PhysicalFormatException e) {
                System.out.println("Invalid binary code");
        }

        Location decodedLoc = null;
        try {
                OpenLRDecoder decoder = new OpenLRDecoder();
                decodedLoc = decoder.decode(decParam, locRef);
            // check validity of decoding result
            if (decodedLoc.isValid()) {
                 return decodedLoc;
            } else {
                return null;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

Anexo IV: Librería “Parser”

Comando CRON

Fichero CRON definido en la maquina de Linux donde esta desplegada la solución:

```
# Edit this file to introduce tasks to be run by cron.

# Each task to run has to be defined through a single line

# indicating with different fields when the task will be run

# and what command to run for the task.

# To define the time you can provide concrete values for

# minute (m), hour (h), day of month (dom), month (mon),

# and day of week (dow) or use '*' in these fields (for 'any').#

# Notice that tasks will be started based on the cron's system

# daemon's notion of time and timezones.

# Output of the crontab jobs (including errors) is sent through

# email to the user the crontab file belongs to (unless redirected).

#
```

```
# For example, you can run a backup of all your user accounts

# at 5 a.m every week with:

# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/

# For more information see the manual pages of crontab(5) and cron(8)

# m h  dom mon dow   command

*/3 * * * *     /home/geoadmin/.ttlive/ttlive-realtime.sh
```

Donde en la ultima línea se especifica el CRON relativo a nuestro sistema. Esta línea indica que se ejecute el comando ttlive-realtime.sh que lanza el JAR encargado del parseo. Se indica que se ejecute el comando cada tres minutos indefinidamente.

Llamada incial del comando CRON

-Método *main* lanzado por el CRON cuando ejecuta la librería "Parser". Este método se encuentra definido en la clase "UpdateTTLiveInfo.java".

```
public static void main (String[] args){

    //Parátros obligatiorios
    String service = "";
    String feed = "";
    String dbConfigFile = "";
    String dbTable = "";
    String dbBinaryTable = "";
    String dbFilterTable = "";
    String dbFinalTable = "";

    try{
        //Par                   tros obligatiorios
        service = args[0];
        feed = args[1];
        dbConfigFile = args[2];
        dbTable = args[3];

        if(service != "" && feed != "" && dbConfigFile != "" &&
               dbTable != ""){
            boolean ok = true;
            try{
                dbBinaryTable = args[6];
            }catch(java.lang.ArrayIndexOutOfBoundsException e1){
                ok = false;
            }
            RTTrafficFeedParser feedParser = new
                RTTrafficFeedParser();
            if(ok){
            //Carga los parámetros de configuración del parser
            feedParser.loadConfigurationFiles(
                    feed,
                    dbConfigFile,
                    dbTable
            );
            //Carga el xml con la informaci e incidencias (nulo ya que
            //queremos que utilice el informado en el feed)
```

```
            feedParser.loadXmlFile(null);

            //Parsea el fichero xml anteriormente cargado y lo carga a
            //base de datos
            feedParser.parseXmlFile();
            }else{
                System.out.println("Falta alguno de los parámetros.");
            }
        }else{
            System.out.println("No se han introducido todos los
                  parámetros necesarios de la aplicación[urlFeed]
                  [dbConfigFile] [dbTable])");
        }

    }catch(java.lang.ArrayIndexOutOfBoundsException e){
        System.out.println("No se han introducido todos los parámetros
                  necesarios de la aplicación[urlFeed] [dbConfigFile]
                  [dbTable])");
    }
}
```

Documento de incidencias

Fragmento del documento xml obtenido a partir del servicio de tráfico en tiempo real ofrecido por TomTom, del cual se obtiene el listado de incidencias. En este fragmento podemos ver los parámetros generales del documento y dos incidencias completas marcadas con la etiqueta <situation>:

```
?xml version="1.0"?>
<d2LogicalModel xmlns="http://datex2.eu/schema/1_0/1_0"
modelBaseVersion="1.0">
  <exchange>
    <supplierIdentification>
      <country>nl</country>
      <nationalIdentifier>TomTom Tele Atlas HD Traffic
Service</nationalIdentifier>
    </supplierIdentification>
  </exchange>
  <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="SituationPublication" lang="en">
    <publicationTime>2012-01-30T17:46:26+01:00</publicationTime>
    <publicationCreator>
      <country>other</country>
      <nationalIdentifier>8ac364ea-fdc5-402e-8580-
6e7a9222455d</nationalIdentifier>
    </publicationCreator>
    <situation id="TTI-8ac364ea-fdc5-402e-8580-6e7a9222455d-
TTL3176130">
      <headerInformation>
        <confidentiality>internalUse</confidentiality>
        <informationStatus>real</informationStatus>
        <urgency>urgent</urgency>
      </headerInformation>
      <situationRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="AbnormalTraffic" id="TTI-8ac364ea-fdc5-402e-8580-
6e7a9222455d-TTL3176130-1">
        <situationRecordCreationTime>2012-01-
30T16:45:26Z</situationRecordCreationTime>
        <situationRecordVersion>1</situationRecordVersion>
```

```
        <situationRecordVersionTime>2012-01-
30T17:46:26+01:00</situationRecordVersionTime>
        <situationRecordFirstSupplierVersionTime>2012-01-
30T17:46:26+01:00</situationRecordFirstSupplierVersionTime>
        <probabilityOfOccurrence>probable</probabilityOfOccurrence>
        <validity>
          <validityStatus>active</validityStatus>
          <validityTimeSpecification>
            <overallStartTime>2012-01-30T16:45:26Z</overallStartTime>
          </validityTimeSpecification>
        </validity>
        <impact>
          <delays>
            <delayTimeValue>98.0</delayTimeValue>
          </delays>
        </impact>
        <generalPublicComment>
          <comment>
            <value lang="en-UK">stationary traffic</value>
          </comment>
        </generalPublicComment>
        <groupOfLocations>
          <locationContainedInGroup xsi:type="Linear">
            <locationExtension>
              <openlr>
                <binary version="3">CwF9tR1mNx55B/4WAKkeS0s=</binary>
              </openlr>
            </locationExtension>
          </locationContainedInGroup>
        </groupOfLocations>
        <situationRecordExtension>
          <alertCEventCode>101</alertCEventCode>
        </situationRecordExtension>
        <abnormalTrafficType>stationaryTraffic</abnormalTrafficType>
        <abnormalTrafficExtension>
          <averageSpeed>10.0</averageSpeed>
        </abnormalTrafficExtension>
      </situationRecord>
    </situation>
    <situation id="TTI-8ac364ea-fdc5-402e-8580-6e7a9222455d-
TTL3176032">
      <headerInformation>
        <confidentiality>internalUse</confidentiality>
        <informationStatus>real</informationStatus>
        <urgency>urgent</urgency>
      </headerInformation>
      <situationRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:type="AbnormalTraffic" id="TTI-8ac364ea-fdc5-402e-8580-
6e7a9222455d-TTL3176032-1">
        <situationRecordCreationTime>2012-01-
30T16:45:26Z</situationRecordCreationTime>
        <situationRecordVersion>1</situationRecordVersion>
        <situationRecordVersionTime>2012-01-
30T17:46:26+01:00</situationRecordVersionTime>
        <situationRecordFirstSupplierVersionTime>2012-01-
30T17:46:26+01:00</situationRecordFirstSupplierVersionTime>
        <probabilityOfOccurrence>probable</probabilityOfOccurrence>
        <validity>
          <validityStatus>active</validityStatus>
          <validityTimeSpecification>
            <overallStartTime>2012-01-30T16:45:26Z</overallStartTime>
```

```
          </validityTimeSpecification>
        </validity>
        <impact>
          <delays>
            <delayTimeValue>146.0</delayTimeValue>
          </delays>
        </impact>
        <generalPublicComment>
          <comment>
            <value lang="en-UK">queuing traffic (with average speeds
Q)</value>
          </comment>
        </generalPublicComment>
        <groupOfLocations>
          <locationContainedInGroup xsi:type="Linear">
            <locationExtension>
              <openlr>
                <binary version="3">C/wmVhqKoRtjBwFsAOAbdVAE</binary>
              </openlr>
            </locationExtension>
          </locationContainedInGroup>
        </groupOfLocations>
        <situationRecordExtension>
          <alertCEventCode>108</alertCEventCode>
        </situationRecordExtension>
        <abnormalTrafficType>queueingTraffic</abnormalTrafficType>
        <abnormalTrafficExtension>
          <averageSpeed>5.0</averageSpeed>
        </abnormalTrafficExtension>
      </situationRecord>
    </situation>
...
```

Anexo V: Servicio SOAP

Uso de Apache Axis 2

A continuación se explica como se ha creado el servicio web utilizando la libreira Apache Axis 2. Para ello se expone la documentación generada para la empresa Geograma:

Crear servicio web Axis2

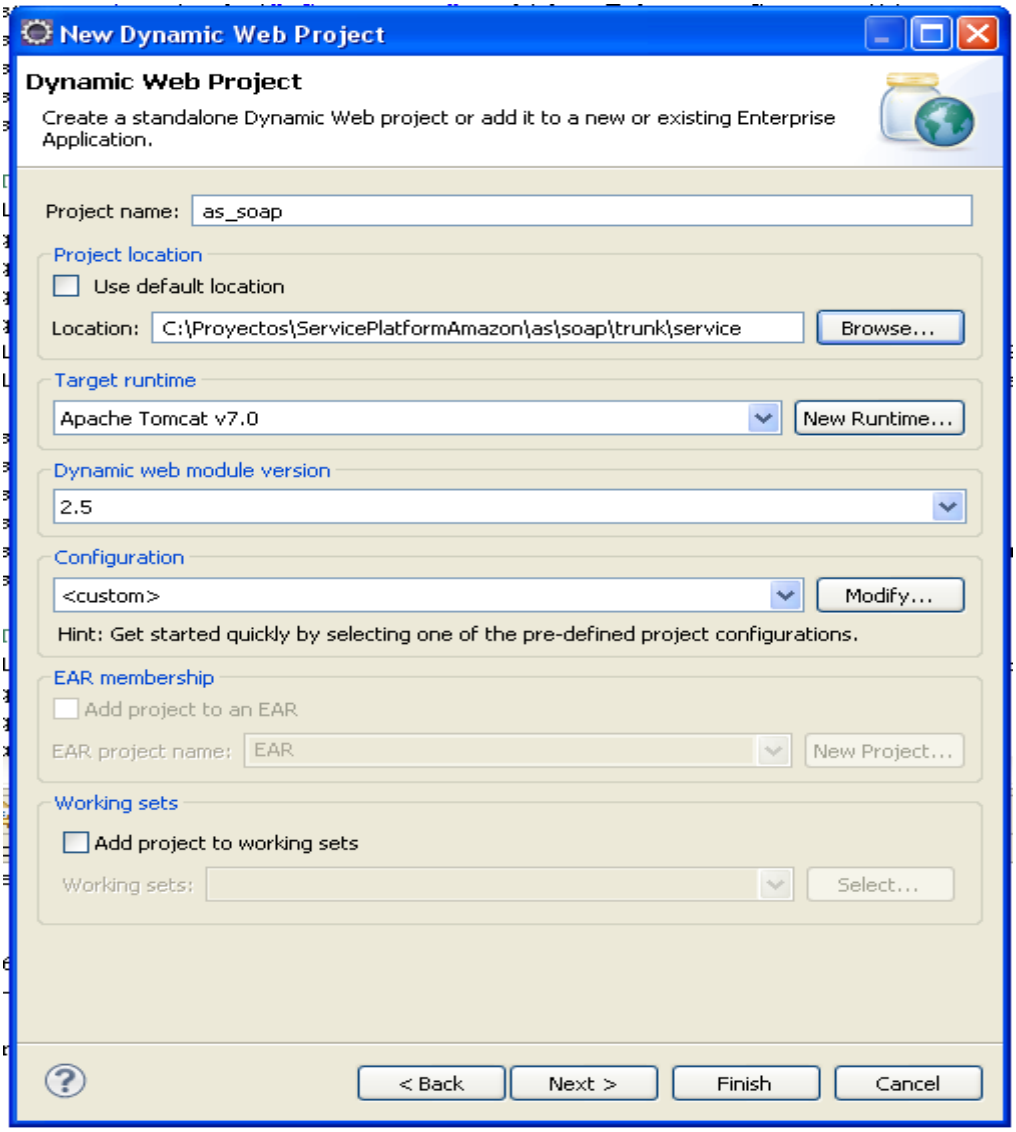
Vamos a crear servicios web con eclipse y un proyecto de prueba para probar el servicio web

Creación del WS

Para mostrar como se debe crear un WB, crearemos uno de ejemplo, en este caso el servicio AS.

-Primero debemos crear un “Dynamic Web Project”, File->New->Other y del listado elegir el proyecto web dinámico.

-En esta ventana elegiremos el modulo 2.5, y yo en mi caso le indico donde quiero que se guarde el código fuente, click en Finalizar:



-Una vez que tenemos el proyecto creado, definiremos nuestro servicio, aquí, cada uno tendrá su funcionalidad y sus clases, pero debemos mantener la misma estructura por lo que explico más o menos como deberíamos hacerlo:

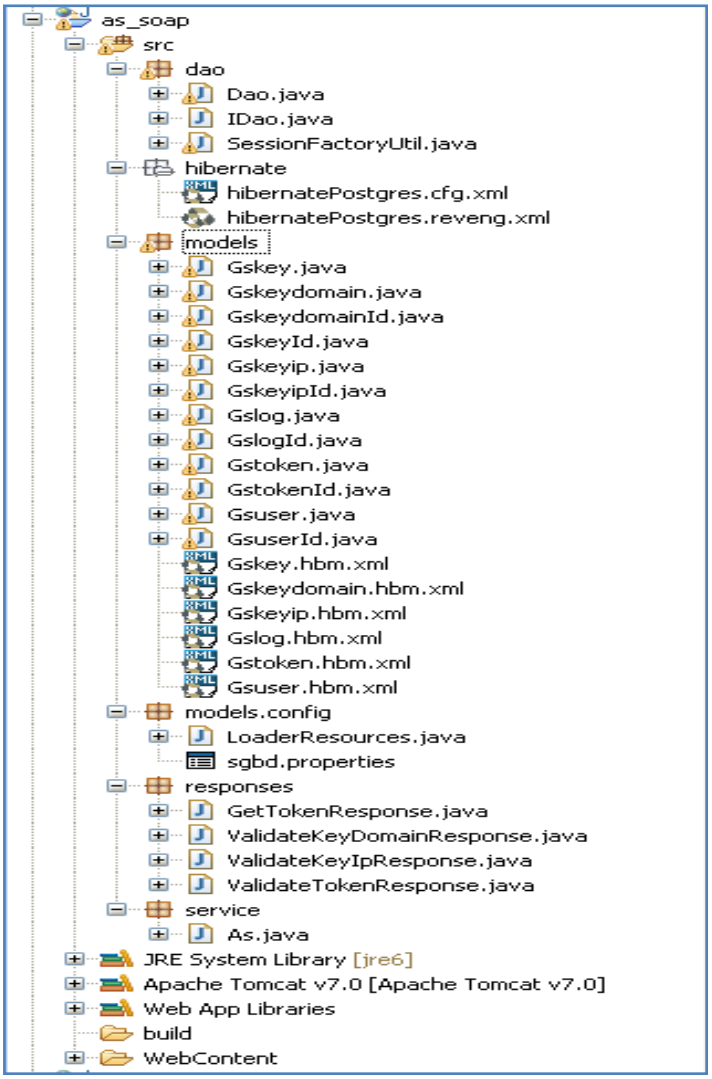
1-Copiar las librerías básicas necesarias para el servicio, como la de logging, hibérnate si necesitamos etc…, en la carpeta webcontent-webinf-lib, eclipse las importará automáticamente al proyecto.

2- Creamos un nuevo paquete donde crearemos la clase principal del servicio web, lo deberíamos llamar “service”, y dentro creamos la clase

3- Creamos otro paquete, que deberíamos llamarlo “responses”, este paquete contendrá los objetos de respuesta del servicio, y dentro crearemos estas clases.

4-Para los accesos a bases de datos, se debe crear otro paquete llamado “dao”, y dentro crear una interface, y la clase que la implementa, de esta forma, el servicio web delegara en estas clases los accesos a la base de datos, xml etc…

La estructura quedaría más o menos así (en este caso hemos añadido también las carpetas relacionadas con las conexiones a la BBDD mediante Hibernate):



**Apuntes a la hora de crear las clases:

-Las clases que sean de respuesta del servicio, debemos generar los getters y setters de todos los atributos que queramos estén visibles para el cliente, si no existen esos getters y setters , el cliente no reconocerá esos atributos.

-Debemos escribir logs, por ejemplo al iniciar una función, al hacer una consulta, cuando ocurra un error…

```
logger.info("Inicio método: "+idAccountProduct+"-"+user+"-"+password);
```

```
logger.error("error: "+e.getMessage());
```

-Para los códigos de error, tanto para los logs como para la respuesta, se puede definir mejor, pero yo elegí unos códigos numéricos, donde por norma general, el 0 es que no existe error, -1 es un error no controlado, -2 parámetros no validos, -3

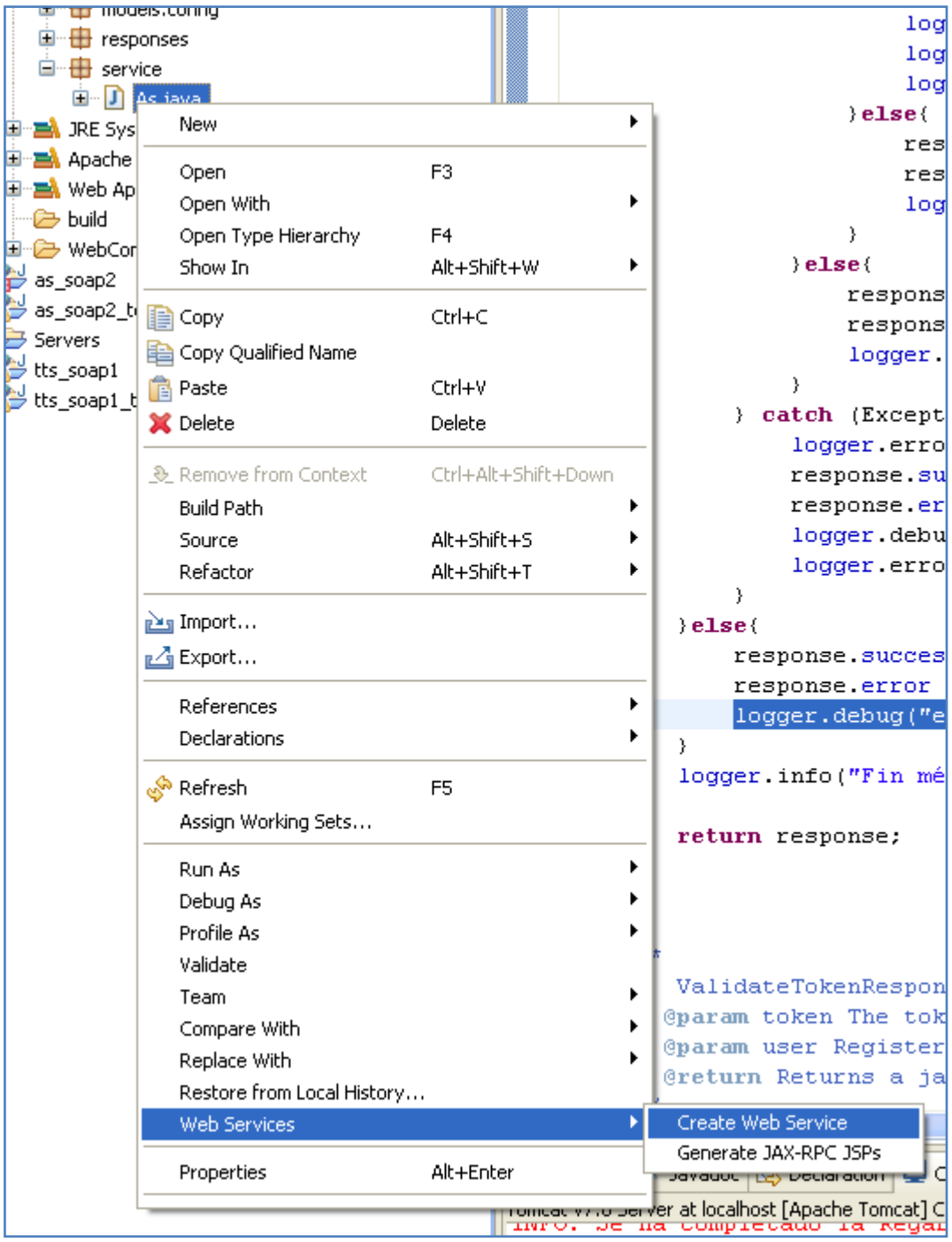
sin permisos para ese usuario. A partir de ahí, se siguen añadiendo códigos, y estos códigos se describen en la función que los devuelve del ws.

-Todas las funciones que proporciona el wb, tiene que responder a unos atributos genéricos, más los propios de cada función, estos son dos:

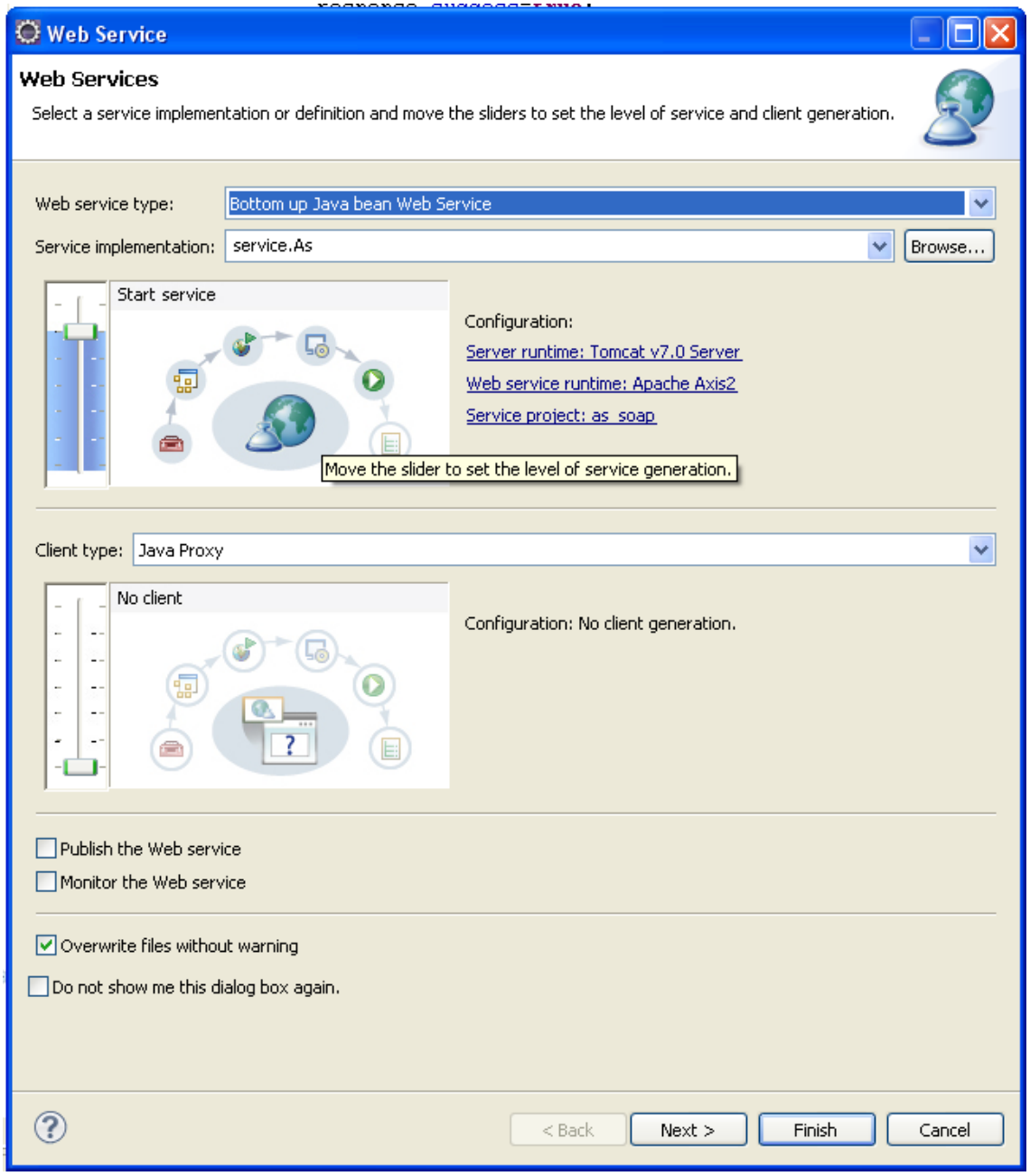
1- success (boolean): indica si todo a transcurrido sin errores. Es decir, devuelve resultados consultados sin errores

2-error(int): el código de error, indica si el usuario no tiene permisos, faltan paramtros, excepción o 0 si todo esta bien etc…

Ahora tocaría crear el ws mediante eclipse, para ello deberemos hacer click derecho encima de la clase principal en este caso As.java:



-En la nueva ventana, deberemos seleccionar Web Service Runtime- > Apache Axis2 y dejar todo lo demás por defecto, click - >Finish:



**Si nos da problemas al crear el webservice, consulta en internet como añadir el Apache Axis2 al tomcat de eclipse.

http://efunctions.wordpress.com/2011/12/19/configurando-apache-axis2-en-eclipse-indigo/

-Después de esto, vemos que eclipse nos ha añadido unas cuantas cosas en el proyecto, en la carpeta webcontent, tenemos una nueva carpeta llamada axis2, y el la cerpeta lib, no ha añadido todas las librerías de axis2.

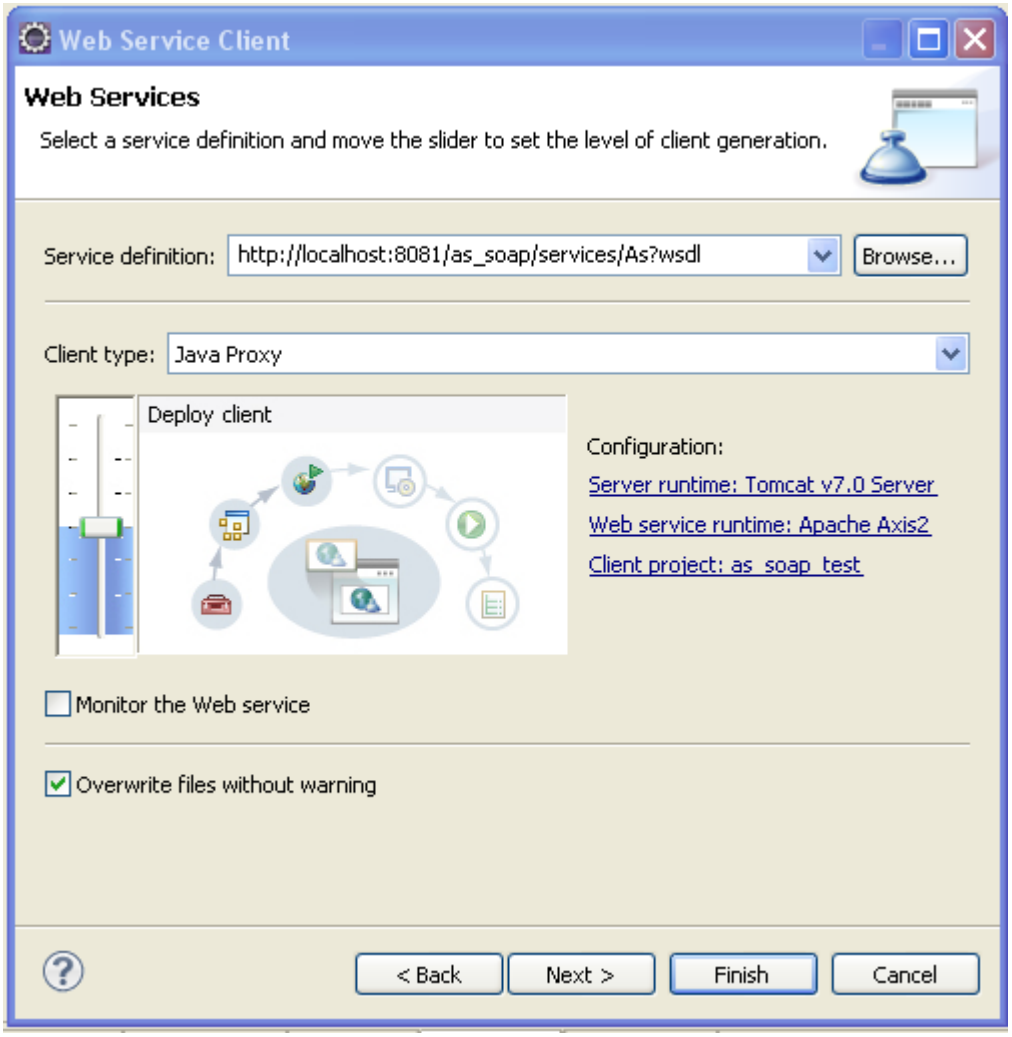
Para probar rápidamente si el servicio está bien, podemos ver el wsdl. Para ello, tenemos que arrancar el tomcat y acceder al wsdl, en mi caso:

http://localhost:8081/as_soap/services/As?wsdl

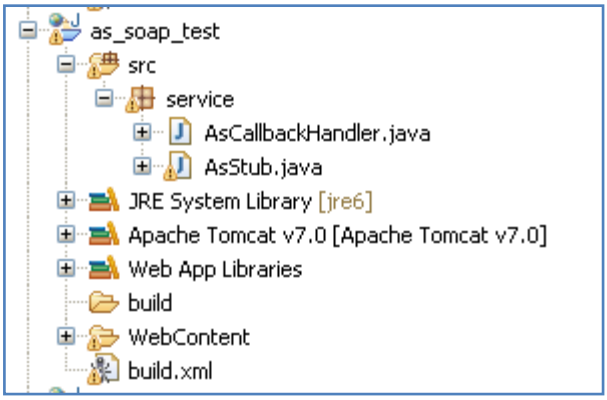
Creamos un cliente

1-File->New->Other y del listado elegimos "web Service Client".

2- En la ventana que nos aparece, debemos indicar donde se encuentra el wsdl, y debemos indicar que utilice Apache Axis2, también un nombre para el proyecto(as_soap_test):



-Click en Finish y eclipse nos crea un nuevo proyecto web dinamico, con las clases que utilizaremos para interactuar con el ws:



-Para acceder al servicio, crear una clase y utilizar la clase de xxxStub.java para acceder al servicio y obtener la respuesta. Todas las clases de respuesta están declaradas en la clase xxxStub

Ejemplo de código:

```
public static void main(String[] args) {
      try{
            AsStub asService = new AsStub();
            //Test getToken function
            GetToken requestGetToken = new GetToken();
            requestGetToken.setIdAccountProduct("1aa");
            requestGetToken.setPassword("endika");
            requestGetToken.setUser("endika");
           GetTokenResponseE responseGetToken =
                      asService.getToken(requestGetToken);
            GetTokenResponse getToken = responseGetToken.get_return();
            System.out.println("GetTokenResponse parameters:" );
            System.out.println("-Success: "+getToken.getSuccess());
            System.out.println("-Error: "+getToken.getError());
            System.out.println("-Token: "+getToken.getToken());
```

```
            System.out.println("-Expire date: "+getToken.getExpire());

            System.out.println();

      }

      catch(Exception oo){

            oo.printStackTrace();

      }

}
```

Conficuraciones de Hibernate utilizadas

Se añade el documento xml utilizado por la librería Hibernate para configurar aspectos como la conexión a la base de datos, el sistema gestor de base de datos mediante el "dialect" de postgres, la configurarión del pool de conexiones C3p0 y los fichero xml de mapeo de entidades o tablas que se utilizaran

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
  <property
name="hibernate.connection.driver_class">org.postgresql.Driver</proper
ty>
  <property name="hibernate.connection.password">A-guinea</property>
  <property
name="hibernate.connection.url">jdbc:postgresql://100.100.100.105:5433
/ttrealtimetraffic</property>
  <property name="hibernate.connection.username">postgres</property>
  <property
name="hibernate.default_catalog">ttrealtimetraffic</property>
  <property name="hibernate.default_schema">gv</property>
  <!-- property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prop
erty-->
  <property
name="hibernate.dialect">org.hibernate.spatial.dialect.postgis.Postgis
Dialect</property>
  <property name="current_session_context_class">thread</property>
  <property name="hibernate.show_sql">false</property>
  <property name="hibernate.format_sql">false</property>
  <property name="use_sql_comments">false</property>
  <property name="hibernate.c3p0.min_size">5</property>
  <property name="hibernate.c3p0.max_size">10</property>
  <property name="hibernate.c3p0.timeout">1000</property>
  <property name="hibernate.c3p0.max_statements">50</property>
  <property name="hibernate.c3p0.idle_test_period">10000</property>
  <mapping resource="models/Realtimetraffic.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

El siguiente documento xml es el mapeo de la tabla RealTimeTraffic de la base de datos en formato xml, que utiliza hibérnate para realizar las consultas a la base de datos.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 30-ene-2013 16:00:42 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="models.Realtimetraffic" table="realtimetraffic">
        <id name="sitid" type="string">
            <column name="sitid" />
            <generator class="assigned" />
        </id>
        <property name="sitconfidentiality" type="string">
            <column name="sitconfidentiality" />
        </property>
        <property name="sitinfstatus" type="string">
            <column name="sitinfstatus" />
        </property>
        <property name="siturgency" type="string">
            <column name="siturgency" />
        </property>
        <property name="sitalertcode" type="java.lang.Integer">
            <column name="sitalertcode" />
        </property>
        <property name="sitver" type="java.lang.Integer">
            <column name="sitver" />
        </property>
        <property name="sitcredate" type="date">
            <column name="sitcredate" length="13" />
        </property>
        <property name="sitfsupver" type="date">
            <column name="sitfsupver" length="13" />
        </property>
        <property name="sitoccurprob" type="string">
            <column name="sitoccurprob" />
        </property>
        <property name="sittype" type="string">
            <column name="sittype" />
        </property>
        <property name="sitvalstatus" type="string">
            <column name="sitvalstatus" />
        </property>
        <property name="sitvaltime" type="date">
            <column name="sitvaltime" length="13" />
        </property>
        <property name="sitvertime" type="string">
            <column name="sitvertime" />
        </property>
        <property name="sitdelaytime" type="java.lang.Double">
            <column name="sitdelaytime" precision="17" scale="17" />
        </property>
        <property name="sittraffictype" type="string">
            <column name="sittraffictype" />
        </property>
        <property name="sitavgspeed" type="java.lang.Double">
            <column name="sitavgspeed" precision="17" scale="17" />
        </property>
        <property name="sitroadeffect" type="string">
```

```
            <column name="sitroadeffect" />
        </property>
        <property name="sitroadmainttype" type="string">
            <column name="sitroadmainttype" />
        </property>
        <property name="sitlocver" type="java.lang.Integer">
            <column name="sitlocver" />
        </property>
        <property name="sitloctype" type="string">
            <column name="sitloctype" />
        </property>
        <property name="sitgeometrydecoded" type="java.lang.Boolean">
            <column name="sitgeometrydecoded" />
        </property>
        <property name="sitlocnumlines" type="java.lang.Integer">
            <column name="sitlocnumlines" />
        </property>
        <property name="theGeom"
type="org.hibernate.spatial.GeometryType">
            <column name="the_geom" />
        </property>
        <property name="sitlocstartname" type="string">
            <column name="sitlocstartname" />
        </property>
        <property name="sitlocendname" type="string">
            <column name="sitlocendname" />
        </property>
    </class>
</hibernate-mapping>
```

Lenguage HQL

Este es lenguaje que utiliza Hibernate para generar las consultas a ejecutar contra la base de datos, lo transforma al lenguaje utilizadao por el gestor de bases de datos y realiza las consultas. Es un lenguaje similar a SQL, pero HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.

Ejemplo de una consulta utilizada por el servicio SOAP utilizando HQL:

```
private ArrayList<Location> getLocations(String types, String bbox)
throws Exception{
        ...
            //Creamos la query en HQL
            String hqlQuery = "FROM Realtimetraffic AS rtt WHERE
rtt.sitgeometrydecoded = true AND intersects(:bbox, rtt.theGeom) =
true";
            logger.debug("Hql query:"+hqlQuery);
            //Obtenemos la session, que seria la conexi□ la BBDD
            Session session =
SessionFactoryUtil.getSessionFactory().openSession();
            session.getTransaction().begin();
            Query query = null;
            //Funcionalidad interna del metodo
            if(hqlQuery.contains(":bbox")){
                String bboxWkt = getWktBbox(bbox);
                Geometry bboxGeom = wktToGeometry(bboxWkt);
                //Preparamos la consulta
                query = session.createQuery(hqlQuery);
                //De esta forma sustituimos el paramtro bbox de la
consulta por la geometria obtenida
```

```
                query.setParameter("bbox", bboxGeom);
            }
            else{
                //Preparamos la consulta
                query = session.createQuery(hqlQuery);
            }
            logger.info("Session loaded");
            //Ejecutamos la consulta
            List<Object> resQuery = query.list();
            //Leemos la respuesta
            for(int i = 0; i<resQuery.size(); i++){
                //Objeto mapeado de la BBDD
                Realtimetraffic location =
(Realtimetraffic)resQuery.get(i);
                //Realizamos la funcionalidad deseada a partir del
obejto devuelto por la BBDD
                Location loc = new Location();
                loc.setGeometry(location.getTheGeom().toString());
                ...
            }
            ...
            //Una vez leido todo, cerramos la session de hibernate,
IMPORTANTE ya que se pueden
            //quedar conexiones basura creadas sin cerrar.
            session.getTransaction().commit();
            session.close();
            logger.debug("Finishing metho");
            return locations;
        }catch(Exception oo){
            logger.error("error: "+oo.getMessage());
            throw oo;
        }
    }
```

A parte de poder utilizar el lenguaje HQL, también se pueden utilizar los objetos java generados para insertar nuevos registros en la BBDD y realizar actualizaciones de registros ya existentes. En nuestro sistema, solo utilizamos Hibernate para la consulta de datos por lo que esta funcionalidad no se utiliza.

Anexo VI: Servicio Servlet

Clase principal

A continuación se muestra un trozo de la clase principal del Servlet. Este servlet recibe un parámetro llamado “request” en el que se indica la funcionalidad que se esta requiriendo, y a continuación llama al método correspondiente para que realice el proceso.

En nuestro servlet, de momento únicamente entiende un tipo de “request”, pero la intención es que en un futuro, accediendo sobre el mismo servicio se puedan realizar diferentes operaciones.

Se puede observar en el fragmento de código como la respuesta tiene formato JSON para facilitar su acceso en la parte cliente.

```
logger.info("Starting method");
```

```
String requestP = request.getParameter("request");
logger.debug("Reading request parameter: "+requestP);
JSONObject resJson = new JSONObject();
if(requestP != "" && requestP != null){
    if(requestP.equals("TRAFFICISSUES")){
        trafficIssues.getTrafficIssues(request, response, soapUrl,
token);
    }else{
        logger.debug("Unknown request parameter: "+requestP);
        resJson.put("success", false);
        JSONObject dataJson = new JSONObject();
        dataJson.put("error", -7);
        resJson.put("data", dataJson);
        if(callback != null && callback != ""){

response.getWriter().print(callback+"("+resJson.toString().replaceAll(
"'", "\\\\'")+")");
        }else{
            response.getWriter().print(resJson);
        }
    }
}else{
    logger.debug("Request parameter null");
    resJson.put("success", false);
    JSONObject dataJson = new JSONObject();
    dataJson.put("error", -6);
    resJson.put("data", dataJson);
    if(callback != null && callback != ""){

response.getWriter().print(callback+"("+resJson.toString().replaceAll(
"'", "\\\\'")+")");
    }else{
        response.getWriter().print(resJson);
    }
}
```

Proceso "trafficIssues" y acceso a SOAP

Como se puede ver en el trozo de código anterior, el servlet delega la funcionalidad en otra clase, "GetTrafficIssues", la que realiza la petición al servicio SOAP y responde encapsulando la respuesta que recibe como JSON.

Utilizando Apache Axis 2, después de haber generado el cliente web explicado en el Anexo V, accedemos utilizando dicho cliente al servicio web SOAP:

```
logger.debug("Parameters: "+username+"-"+idAccountProduct+"-
"+callback+"-"+types+"-"+bbox);
//Conecting to as soap service
TtsStub ttsService = new TtsStub(soapUrl);
TtsStub.TrafficIssues requestIssues = new TtsStub.TrafficIssues();
requestIssues.setIdAccountProduct(idAccountProduct);
requestIssues.setToken(token);
requestIssues.setUsername(username);
requestIssues.setIncidenceTypes(types);
requestIssues.setBbox(bbox);
TrafficIssuesResponseE responseIssues =
ttsService.trafficIssues(requestIssues);
TrafficIssuesResponse infoResponse = responseIssues.get_return();
```

Una vez tenemos la respuesta, lo que hacemos es recorrer el listado de incidencias para añadirlas al JSON de respuesta:

```
//read global parameters for the response
resJson.put("success", infoResponse.getSuccess());
JSONObject dataJson = new JSONObject();
dataJson.put("error", infoResponse.getError());


Location[] locations = infoResponse.getIssues();
JSONArray locationsArray = new JSONArray();
if(locations != null){
    for(int j=0; j<locations.length; j++){
        JSONObject locationsJson = new JSONObject();
        Location location = locations[j];
        if(location != null){
            locationsJson.put("sitAvgSpeed",
                        location.getSitAvgSpeed());
            locationsJson.put("sitAlertCode",
                        location.getSitAlertCode());
            locationsJson.put("sitDelayTime",
                        location.getSitDelayTime());
      . . .
            locationsArray.put(locationsJson);
        }
    }
}
dataJson.put("locations", locationsArray);

resJson.put("data", dataJson);
```

Finalmente comprobamos si tenemos que añadirle una función de callback a la respuesta. Esto es debido a JSONP:

```
logger.debug("Sending response");
String callback = request.getParameter("callback");
if(callback != null && callback != ""){

response.getWriter().print(callback+"("+resJson.toString().replaceAll(
"'", "\\\\'")+")");
}
else{
    response.getWriter().print(resJson);
}
logger.info("Finishing method");
```

Anexo VII: API Javascript

TomTom Traffic API

Javascript API for the TomTom traffic service

- Sample usage:

```
//Instantiate the API service - "GEOSERVICIOS.tts"
GEOSERVICIOS.tts.getTrafficIssues({
    username: "thename",
    key: "key",
    idaccountproduct: "1aa",
    callback: thecallback,
    types: "slowTraffic,works",
    bbox: "theBbox"
});

function thecallback(response){
    //Do your work here
}
```

Summary

TomTom Traffic API	Javascript API for the TomTom traffic service
FUNCTIONS	
getTrafficIssues	Returns a list of geographical names of an administration hierarchical level of a country.

FUNCTIONS

getTrafficIssues

```
getTrafficIssues: function(options)
```

Returns a list of geographical names of an administration hierarchical level of a country. You can use filters too:

Parameters

idaccountproduct	{String} The identifier for the hired product.
username	{String} The name of a registered user for the product.
key	{String} A valid key for this users IP and Domain.
callback	{Function} The function used for the callback
types	{String} The types of incidences we want to receive, separated by commas(optional)
bbox	{String} The bounding box that contains the incidences(optional)

Returns {JSON}

Parameters

success	{boolean} Indicates if the request has been successfully finished
data	{JSON} The information returned from the web service

Sample usage

```
GEOSERVICIOS.tts.getTrafficIssues({
    username: "thename",
    key: "key",
    idaccountproduct: "1aa",
    callback: thecallback,
    types: "slowTraffic,works",
    bbox: "theBbox"
});

function thecallback(response){
    //Do your work here
}
```

JSONP y cross-domain

Ya se ha comentado que JSONP es una técnica para “saltarse” el cross-domain, la restricción que viene impuesta en los navegadores para no poder acceder a información alojada en otros contextos de aplicación diferentes a los de la pagina web, por cuestiones de seguridad.

Para poder utilizar esta técnica es necesario añadir en los servicios una función de callback que será enviada por el cliente como parámetro, y que el formato de la respuesta venga en formato JSON:

```
String callback = request.getParameter("callback");
if(callback != null && callback != ""){

response.getWriter().print(callback+"("+resJson.toString().replaceAll(
"'", "\\\\'")+")");
}
else{
    response.getWriter().print(resJson);
}
```

En la parte cliente, a la hora de hacer la petición no utilizaremos AJAX, si no que incrustaremos en la cabecera del documento web un script con la url apuntando al servicio:

```
//Generamos un identificador univico
var random = Math.floor((Math.random()*100000)+1);
var idRequest = "tts_" + random ;

//Declaramos la function de callback
window[idRequest] = function(response){
    var script = document.getElementById(idRequest);
    var head = document.getElementsByTagName("head").item(0);
    if (script != null) {
        head.removeChild(script);
    }
    //En este punto tengo acceso a la respuesta "response"
};

//Creamos la URL indicando la funcion de callback que acabamos de
definir
url =
"URL_AL_SERVICIO?[PARAMETROS_PETICION]&callback=window."+idRequest;

// Añamos la petición la cabecera
var source = url;
var nodoTool = document.createElement('script');
with (nodoTool) {
    id = idRequest;
    type = 'text/javascript';
    src = source;
}
document.getElementsByTagName("head")[0].appendChild(nodoTool);
```