2009

Universidad Carlos III de Madrid

Ingeniería Superior de Telecomunicaciones

Alejandro Cornejo Benedito

HERRAMIENTA WEB PARA LA SUPERVISIÓN DE UN PEAJE DE CARRETERAS

Descripción técnica y funcional del desarrollo de una aplicación Web para la supervisión y monitorización del sistema de un peaje de carreteras.

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES

1 INTRODUCCIÓN

El Proyecto de Fin de Carrera que aquí se presenta consiste en el desarrollo de una aplicación destinada a la Supervisión y Monitorización de un peaje de carreteras basado en tecnologías web.

Dicha aplicación incluirá por una parte diversas funcionalidades destinadas a la monitorización en tiempo real de los distintos elementos y ubicaciones del peaje, así como funcionalidades destinadas a acceder y mostrar al usuario ciertos datos almacenados en la Base de Datos del sistema. Por último, la aplicación también consta de un módulo destinado a la configuración del sistema.

En definitiva, se pretende aglutinar en una única aplicación todas aquellas funciones necesarias para mantener el control y la supervisión sobre un peaje de carreteras que, por definición, es un sistema distribuido con entidades distantes en el espacio.

1.1. JUSTIFICACIÓN Y OBJETIVOS

Se ha escogido una aplicación web con el objetivo de facilitar el trabajo al conjunto de controladores y supervisores del peaje de carreteras que hasta ahora disponían de distintas herramientas más localizadas y dirigidas exclusivamente a ciertas tareas de control y a supervisar únicamente ciertos elementos del peaje. Con esta aplicación web se pretende poner al servicio del personal a cargo de la supervisión del peaje una única aplicación accesible desde cualquier ubicación y que aglutine todas aquellas tareas que antes requerían la ejecución de diversos procesos.

Esta aplicación permitirá monitorizar en tiempo real todas las ubicaciones del peaje de carreteras de modo que a través de las distintas funcionalidades que proporciona, se podrán monitorizar todas las alarmas que en un determinado instante se estén produciendo en cualquier lugar del peaje. Esta monitorización de alarmas proporcionada por una única aplicación es extremadamente importante para el correcto diagnóstico, posterior corrección, prevención y ulteriores labores de mantenimiento de los distintos elementos que componen el peaje. Estas ventajas se traducen automáticamente en una disminución muy considerable de los costes asociados al mantenimiento del peaje así como un aumento en los beneficios obtenidos de la explotación de la concesión.

Para que todo esto sea posible y funcione correctamente, ha sido necesario montar una arquitectura completa de comunicaciones entre las distintas entidades involucradas que permita la transferencia de mensajes y el almacenamiento de toda la información del sistema en una entidad jerárquicamente superior al resto (llamada Centro de Control) de donde la aplicación web extrae en tiempo real toda la información que se le solicita.

Se ha tratado en todo momento de desarrollar una aplicación que sea altamente configurable de modo que pueda exportarse a peajes con distintas características y exigencias y sea el personal a cargo de la Supervisión del peaje quienes adapten la aplicación a sus propias necesidades.

2 ARQUITECTURA BÁSICA DE UN SISTEMA DE PEAJE

En este apartado se pretende describir someramente la arquitectura que subyace a un peaje de carreteras para que pueda entenderse posteriormente el funcionamiento y el objetivo que se persigue con la aplicación web de Supervisión.

Un peaje de carreteras se compone de distintas entidades jerárquicamente organizadas que se pueden resumir en tres niveles a saber, **vías**, **estaciones** y **centro de control**.

La Figura 1 representa esquemáticamente la estructura jerárquica básica de un sistema de peaje.

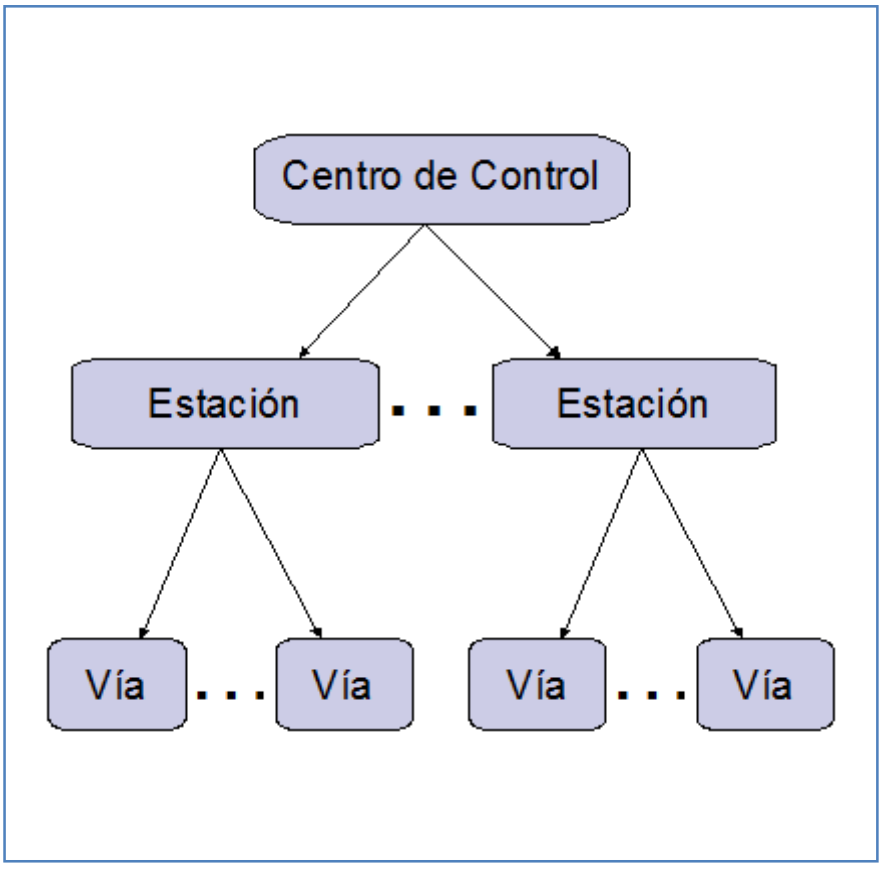

Figura 1: Estructura jerárquica de un peaje

2.1.VÍAS

Dentro de la organización jerárquica de un peaje las vías corresponden al nivel inferior. Por las vías transitan los vehículos que hacen uso del peaje de carretera. Las vías están compuestas por una serie de elementos que son controlados por ciertos procesos que corren en unas máquinas que están físicamente ubicadas en cada una de ellas. Tales elementos pueden ser barreras, semáforos, pisones, antenas, bases de datos, discos duros, etc…

Las vías pueden abrirse para su uso fundamentalmente en tres modos a saber, **manual** cuando hay un peajista físicamente ubicado en la vía para cobrar el importe del tránsito, **automático** cuando existe una máquina destinada a realizar dicha labor y en **telepeaje** destinado a aquellos usuarios frecuentes de la autopista que portan en sus vehículos tarjetas asociadas a saldos que pueden ser leídas por antenas ubicadas en la vía.

2.2.ESTACIONES

Un conjunto de vías está controlado por una estación que desde el punto de vista funcional puede considerarse un proceso corriendo a un nivel jerárquicamente superior y que controla y centraliza los mensajes provenientes de las vías que tiene por debajo.

Hasta ahora la supervisión y actuaciones sobre las vías se tenían que realizar a través de un software instalado en la estación que se conectaba a aquellas por medio de sockets. Este funcionamiento requiere que los controladores del sistema de peaje tengan que correr la aplicación de estación para monitorizar las vías y desconectarse, desplazarse y volverse a conectar en otra estación distinta cuando se quiere monitorizar las vías de otra estación.

Con la aplicación web de Supervisión que se está presentando, se pretende centralizar en una sola aplicación accesible desde cualquier máquina que tenga conectividad con el servidor web todas las labores que antes requerían acudir físicamente a la estación que se quería monitorizar.

2.3. CENTRO DE CONTROL

El Centro de Control es la entidad jerárquicamente superior del sistema de peaje. Se trata de una máquina que recoge y centraliza la información proveniente de todas las vías.

En el Centro de Control hay unos controladores que dan soporte remoto a los peajistas que están en las vías. Además estos controladores están encargados de emitir órdenes de mantenimiento al personal correspondiente cuando detectan alguna anomalía en el funcionamiento de algún elemento o equipo del sistema. La rápida detección de estos problemas es fundamental para que sean reparados lo antes posible, además de para establecer un diagnóstico certero y en consecuencia establecer medidas preventivas que protejan al sistema de posibles fallos en el futuro de manera que el impacto sobre el servicio sea mínimo.

A este nivel debe desplegarse la aplicación web de Supervisión. Desde aquí es posible monitorizar absolutamente todas las vías del peaje merced a una arquitectura de comunicaciones que posibilita la trasferencias de mensajes desde los niveles inferiores a los superiores para el posterior almacenamiento de dicha información y que sirve para nutrir de información en tiempo real a la aplicación web de Supervisión.

Más adelante se describe cual es el funcionamiento y la arquitectura de transferencia de mensajes que posibilita la supervisión centralizada de todo el sistema.

3 ESTADO DEL ARTE

En este apartado se va a explicar cuál es el avance real que supone el desarrollo de una aplicación con las características expuestas en la introducción con respecto a la solución desplegada hasta este momento. Para ello se va a describir la solución que había implementada hasta este momento en el sistema de peaje y así exponer posteriormente las ventajas que supone el despliegue de la Aplicación web de Supervisión.

De este modo, este apartado supone en cierta manera la prolongación del apartado 1.1 titulado *Justificación y Objetivos*, así como la puesta en antecedentes que sirva para entender con qué motivación exacta está desarrollada esta aplicación, así como cuáles son los objetivos que persigue y los requisitos que debe reunir.

Con estos mimbres, en el apartado 4 titulado *Descripción Técnica*, se justificará la arquitectura elegida en base a la mejor opción que permita cumplir con las exigencias impuestas por la explotación de esta aplicación.

3.1. TERMINAL TUTOR DEL SISTEMA DE PEAJE

El terminal tutor es una aplicación desplegada a nivel de estación desarrollada con el objetivo de monitorizar ciertos elementos de las vías localizadas en dicha estación. Adolece de grandes limitaciones que serán explicadas a continuación.

Concretamente el terminal tutor es capaz de mostrar en tiempo real la siguiente información:

- Las alarmas de los periféricos de las vías.
- Estados de vía: abierta, cerrada y modo de operación.

Las vías desplegadas en la estación se conectan con esta mediante sockets y envían ristras de bytes que el tutor es capaz de decodificar. Este modelo de comunicaciones limita enormemente la cantidad de información susceptible de ser enviada en un formato de mensaje que se limita a mantener flags activos o inhibidos en función del estado de ciertos elementos.

Más adelante veremos cómo la aplicación web de Supervisión se nutre de mensajes XML difundidos a través de la arquitectura de comunicaciones del peaje diseñados al efecto de cubrir la totalidad de la información que la vía es capaz de generar, incluso dejando abiertas posibilidades a futuras ampliaciones.

Existe un terminal tutor situado en cada estación para la gestión de todas las vías que componen esa estación. Se trata, por tanto, de una aplicación con un ámbito de operación limitado únicamente a las vías que pertenecen a dicha estación. Con la aplicación web de Supervisión se pretende dar un paso más y ampliar el ámbito de operación a todo el peaje. Para ello se ha diseñado una estructura jerárquica de comunicaciones que se detalla posteriormente y que permite el flujo de mensajes a través de todos los niveles del peaje hasta llegar a su nivel jerárquico máximo, a saber, el Centro de Control.

El terminal tutor es, además, una aplicación que requiere ser ejecutada en local y no permite su ejecución en remoto salvo que se abra un terminal VNC con conexión a la máquina en la que está instalado, lo cual impone grandes limitaciones de operatividad. La aplicación de Supervisión, al ser una aplicación web, pretende romper estas limitaciones de ubicuidad y permite ser accesible y estar operativa desde cualquier localización del peaje en condiciones normales de conectividad.

El terminal tutor dispone de un motor de base de datos local que utiliza para hacer la validación de usuario, así como para registrar todas sus actuaciones y la información de las incidencias recibidas. Esto da idea nuevamente de las restricciones de localización que tiene esta aplicación. La aplicación web de Supervisión, accede a unas Bases de Datos que mantienen información de todo el peaje. Todos los eventos y alarmas que tienen lugar en cualquier localización del sistema de peaje son transmitidos desde las vías hasta el Centro de Control y almacenados en las Bases de Datos para su posterior consulta y análisis.

La concentración en una única Base de Datos de toda la información generada en cualquier lugar del peaje, facilita el análisis de las causa de error y, en consecuencia, permite que se establezcan políticas de prevención más acertadas. Además, la rápida detección de los errores producidos en cualquier lugar del peaje permite que las labores de mantenimiento se agilicen, lo cual redunda en un importante recorte en las pérdidas que se puedan derivar de la prolongada inoperatividad de las vías.

Todo ello sin contar con el importante recorte en el personal de supervisión que viene derivado precisamente la capacidad de la aplicación web de Supervisión y la arquitectura subyacente de aglutinar toda la información del peaje en un único punto de acceso.

4 DESCRIPCIÓN TÉCNICA

4.1. REQUISITOS DE EXPLOTACIÓN

Se trata de una aplicación crítica puesto que gestiona el funcionamiento y el tratamiento de la información que llega al Centro de Control de Peajes de una Autopista. Ello requiere que sea una aplicación de alta disponibilidad y servicio 24x7.

La aplicación debe tener varias funcionalidades distintas accesibles desde un mismo interfaz. Debe estar basada en una arquitectura que favorezca una fácil y ágil interrelación entre los diferentes módulos, así como un rápido intercambio de información con el objetivo de refrescar dicha información al usuario con la mayor celeridad posible.

Se debe garantizar el acceso a las BBDD, a los sistemas de control de peaje, y al envío y recepción de información crítica, por lo que la elección de una arquitectura debe basarse en la fiabilidad y la capacidad de asumir todas las necesidades impuestas.

Un Aplicación Web proporciona todas estas ventajas y muchas más:

- Es ubicua. Es decir, cualquier operario puede acceder a dicha aplicación sin más que tener un ordenador con conectividad al servidor y un navegador.
- Permite unificar en una sola aplicación bajo un mismo interfaz todas las funcionalidades que anteriormente estaban dispersas en distintas aplicaciones a las cuales había que acceder desde diferentes puestos.
- El navegador es un cliente ligero y no requiere una instalación previa en las máquinas desde las que se quiere acceder.
- La arquitectura escogida detallada a continuación permite un rápido acceso tanto a las Bases de Datos como al Cliente en Tiempo Real y una rápida interrelación entre todos los módulos.
- Favorece el trabajo en equipo puesto que se le dota de modularidad y no dificulta posteriores ampliaciones.

Estas son sólo algunas de las ventajas que ofrece la aplicación web. A continuación se describe y se explica la arquitectura escogida y cuáles son sus bondades.

4.2.COMPARATIVA ENTRE 3 POSIBLES LENGUAJES PARA DESARROLLO WEB

Una vez decidido que una aplicación web satisface las necesidades y requisitos necesarios para una aplicación de las características observadas en el apartado anterior, el siguiente paso consiste en elegir, de entre las posibles tecnologías existentes, las que mejor se adapten a nuestras necesidades.

Lo primero que resulta necesario escoger en el momento necesario de enfrentarse e un desarrollo web es el lenguaje de programación que se va a utilizar. A continuación se muestra una tabla comparativa entre J2EE, ASP.NET y PHP, que son los lenguajes de programación más empleados en desarrollo web.

Criterio	J2EE	ASP.NET	PHP
Sintaxis	Sencilla	Visual Basic	PHP aún usa '::' y '->' y algunas funciones podrían ser usadas dentro de los objetos y no como procedimientos
Curva de aprendizaje	Complicado	Sencillo	Sencillo
Velocidad de desarrollo	J2EE es el más lento.	Es el más rápido, debido a la cantidad de componentes que tiene que te hacen todo el trabajo	Es rápido si se usa algún framework
Plataforma	Trabaja bien en cualquier plataforma	Windows	Trabaja mejor en LAMP
IDE	Eclipse	Visual Studio. De coste elevado	Se puede usar Eclipse
Soporte orientado a objetos	Alto	Alto	Medio
Seguridad	Seguro	Fallos de seguridad debidos a Windows.	Mala fama

Rendimiento	Pesado	Pesado	Ligero
Servidor Web	Versiones comerciales y open source	IIS	Versiones comerciales y open source
Librerías y frameworks	La mayoría gratuitas y open source	Propietario	La mayoría gratuitas y open source
Soporte y comunidad	Grupos independientes.	Foros, grupos de usuarios y comunidades de desarrolladores manejados por Microsoft	Grupos independientes.
Coste	Herramientas gratuitas y de pago	Licencias caras	Gratuita

Se requiere un framework de código abierto, gratuito y con un amplio soporte para el desarrollo. El PHP se queda corto para las necesidades de este proyecto, mientras que .NET no cumple ninguno de estos requisitos, por lo que el lenguaje escogido es J2EE.

4.3.JUSTIFICACIÓN DE LA ARQUITECTURA ELEGIDA

4.3.1. MODELO EN N CAPAS

El objetivo principal de la arquitectura es separar, de la forma más limpia posible, las distintas capas de desarrollo, con especial atención a permitir un modelo de domino limpio y a la facilidad de mantenimiento y evolución de las aplicaciones. Otro elemento importante es la facilidad del despliegue.

Se ha pensado en un tipo de arquitectura que permita definir las diferentes partes del proyecto como elementos independientes con una separación de responsabilidades entre ellos (arquitectura en n capas) y que sea escalable, permitiendo un aumento del uso sin que disminuya su rendimiento.

Para lograr esto se eligió el patrón MVC (Modelo-Vista-Controlador) que permite una separación limpia entre las distintas capas de una aplicación.

4.3.2. CAPA DE PRESENTACIÓN

Para la primera de las capas, la de presentación, se ha pensado sobre todo en la flexibilidad y facilidad a la hora de elaborar pantallas, gestión de elementos de las mismas, validaciones, gestión de errores, mapeo de los elementos con sus correspondientes clases de persistencia, e inclusión de elementos complejos como menús, árboles, ajax, etc.

La capa de presentación recoge la entrada del usuario, presenta los datos, controla la navegación por las páginas y delega la entrada del usuario a la capa de la lógica de negocio. La capa de presentación también puede validar la entrada del usuario y mantener el estado de sesión de la aplicación.

Se pueden incluir en el apartado de la capa de presentación, una serie de utilidades que permitan un mejor uso de las funcionalidades propuestas, por lo que se ofrece la utilización de librerías

AJAX como mejora en los tiempos de respuesta ante peticiones al servidor (peticiones de recarga de página, peticiones de respuesta ante eventos de la misma, etc.), ya que permite la gestión de los eventos en background, lo que proporciona una gestión más amigable del entorno Web.

Esta arquitectura permite la utilización tanto de **JSP's** como de **JSTL**, para la capa de presentación, y de la inclusión de librerías de etiquetas **TLD** (Tags-Libs) que ya se hayan utilizado o definido anteriormente.

Un poco más abajo, en este mismo apartado, se enumeran y describen las tecnologías empleadas tanto de esta capa como en el resto.

4.3.3. CAPA DE NEGOCIO

En la capa de Negocio se optó por una solución basada en servicios (no necesariamente servicios web, aunque permitiendo su integración de forma limpia) que trabajaban contra un modelo de dominio limpio. La persistencia de las clases se sustenta en **DAO's** (Objetos de Acceso a Datos), manteniendo aislada la capa de persistencia de la capa de negocio. Tanto los servicios como los DAO's así como el propio modelo son realmente **POJO's** (clases simples de Java), con la simplicidad que conllevan y sin dependencias reales con ningún framework concreto. Para realizar esta integración se ha elegido **Spring.**

Spring resulta el framework más indicado para enlazar la capa de negocio con Hibernate, que es la tecnología de persistencia escogida, cuya elección se justifica en el siguiente punto. Spring integra gran cantidad de API's y templates que permiten un fácil y amplio manejo de las posibilidades que brinda Hibernate sustentado sobre una muy sencilla labor de configuración.

Por otra parte, para la integración de la capa de presentación con la de negocio, se ha empleado Struts puesto que resulta un framework de muy fácil manejo para el intercambio de datos entre ambas capas, así como para definir el flujo de acciones que gestiona dicho intercambio y otras ventajas que se analizarán más adelante.

4.3.4. CAPA DE PERSISTENCIA

La capa de persistencia se encarga de gestionar y obtener la información de los diferentes objetos definidos en la Base de Datos. Si se utilizase un conjunto de clases definidas a medida sería necesario definir métodos de acceso, conexión, etc... Sin embargo se plantea la posibilidad de utilizar un

framework ya existente para ello que se encarga de gestionar tanto la conectividad hacia los diferentes motores de BBDD como la utilización de la información obtenida de los mismos.

A continuación se expone una tabla comparativa entre diferentes alternativas para esta capa indicando cuáles son las ventajas y desventajas de cada una de ellas.

Tecnología	Ventajas	Desventajas
JDBC		• Código específico de Base de Datos • Acceso directo a conexiones y transacciones • Transacciones a un único DataSource • El desarrollador Java debe saber tambien otro lenguaje (SQL) • El código se hace repetitivo
iBatis	• Simplicidad • Curva de aprendizaje • Control completo sobre los SQL • Rendimiento • Flexibilidad • Permite mapeo directo	• No es un ORM (Object Relational Mapping) • El desarrollador debe tener altos conocimientos de SQL • Transacciones restringidas a un DataSource • Baja portabilidad
EJB (ORM)	• Provee servicios de forma transparente • Transacciones, Seguridad, Conexiones a BD • Rendimiento, si se llega a dominar (cache) • Clustering • RMI • Muchos servidores de aplicaciones importantes lo soportan	• Curva de aprendizaje • Necesita un contenedor de aplicaciones • Intrusivo: las clases a persistir deben implementar interfaces EJB • Rendimiento bajo • No soporta Herencia ni relaciones entre clases • EJB 2 introduce CMR para asociaciones • Portabilidad • Testing difícil y lento
Hibernate (ORM)	• No intrusivo (estilo POJO)	• No es estándar

	• Muy buena documentación • Muchos usuarios • Transacciones, caché, asociaciones, poliformismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching… • Potente lenguaje de Consulta (HQL) • Fácil testeo	
JPA (ORM)	• Testing • Simplicidad • Facilidad de aprendizaje • Transparencia: POJOs • Herencia, poliformismo)	• Descripción de la Base de Datos en el código

Dadas las posibilidades de elección entre las distintas tecnologías posibles, la primera consideración es que para un proyecto grande, modular y con enormes posibilidades de ampliación, se necesita una tecnología ORM relacional que no requiera emplear SQL nativo y que al mismo tiempo ofrezca una gran versatilidad a la hora de desarrollar los distintos métodos de acceso a las Bases de Datos.

La consideración previa excluye directamente las dos primeras opciones propuestas: tanto iBatis como JDBC. De las tres restantes, el framework que ofrece mayor versatilidad, posibilidades de desarrollo y requiere menos código es **Hibernate**.

Hibernate ofrece un gran abanico de posibilidades de ejecución y diseño relacional a través de las directivas que ofrece para el mapeo de sus tablas en elementos Java, así como de relaciones de parentesco entre ellas. Esto unido a la potente unión que supone utilizar **Hibernate** y **Spring** juntos y a la sencillez de su manejo, resulta definitivo para la elección de esta dupla para las capas de negocio y persistencia.

4.4.DIAGRAMA DE LA ARQUITECTURA

La estructura de los componentes que conforman la arquitectura se ajusta a la que se representa la Figura 20.

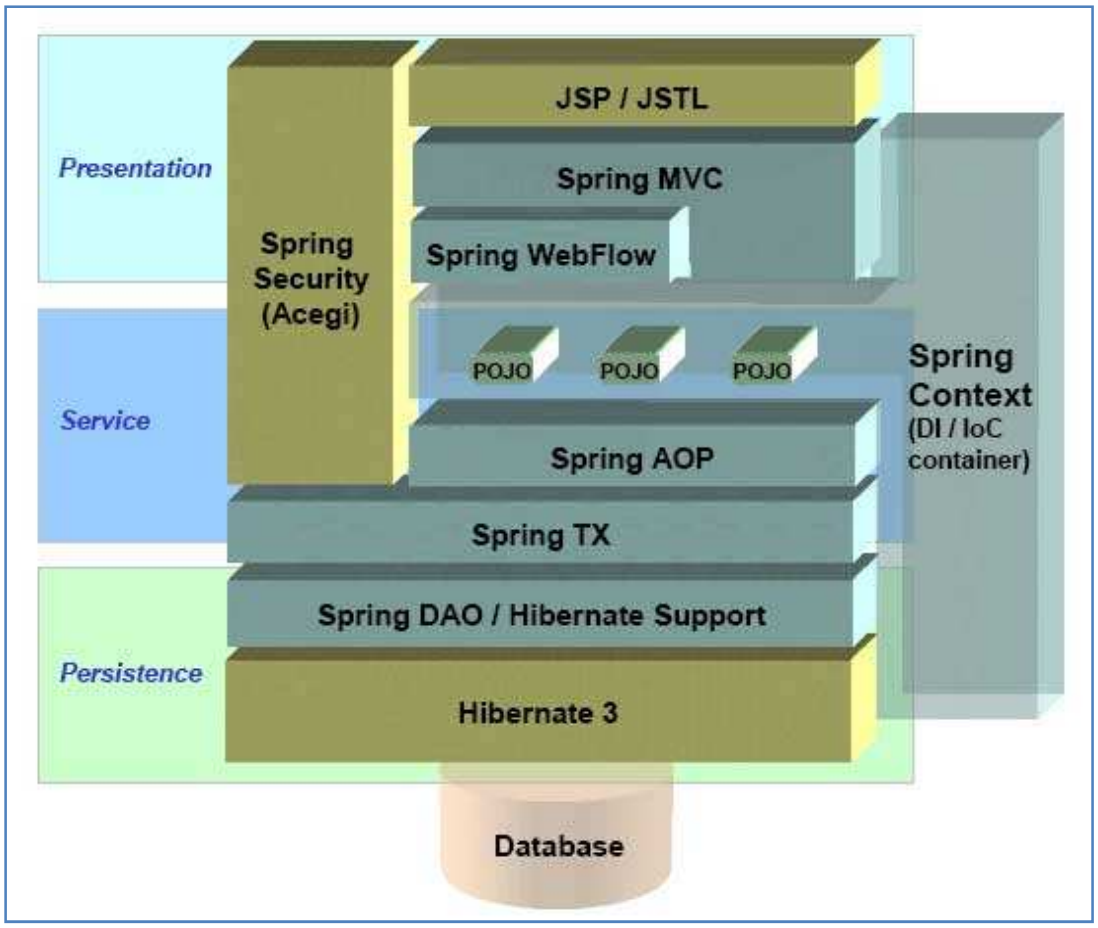

Figura 2: Modelo de capas

4.5.FRAMEWORKS EMPLEADOS

A continuación se enumeran cada una de las tecnologías empleadas en el desarrollo de la Aplicación Web dando una breve descripción para cada una de ellas, explicando brevemente cual ha sido la utilidad o desempeño de cada una de ellas en la Aplicación.

4.6.1 JSP, JAVASCRIPT, CSS JSTL, TLD Y TILES

En la Capa de Presentación se utilizan estas tecnologías para construir el interfaz de usuario, presentarle a este los datos recogidos y proporcionarle un interfaz de control que le permita realizar ciertas acciones o solicitar cierta información.

Las **JSP (Java Server Pages)** son páginas web generadas de forma dinámica por el Servidor Web. A partir de la **JSP** el Servidor genera una página **html** construida con datos generados dinámicamente que le envía al navegador del cliente para que este se encargue de mostrárselo al usuario.

Para construir las **JSP**, se utilizan etiquetas de librería llamadas **TLD (Tag Library Descriptors)** bajo las cuales subyace la ejecución de código Java. El framework **JSTL (Java Server Tag Libraries)** proporciona un amplio abanico de etiquetas que facilitan enormemente la labor al desarrollador de la aplicación a la hora de construir las JSP.

Además de estas etiquetas predefinidas en el framework **JSTL**, las **TLD's** también pueden ser definidas por el usuario.

Además para facilitar la maquetación de las páginas web mostradas al usuario, se ha hecho el uso de **Struts Tiles**. Se trata de construir las páginas web a base de templates cada uno de los cuales genera una parte de las misma: cabecera, pie, menú, cuerpo, javascript, css ….

La definición de los **Tiles** se realiza en ficheros XML y asigna alias que luego pueden utilizarse para configurar los actions de struts.

Además se ha hecho uso de **javascript** a la hora de programar toda aquella funcionalidad de ejecución en la parte cliente como por ejemplo validación y control de campos de formularios, cualquier tipo de actuación sobre los datos introducidos por el usuario previa a la invocación del **Action de**

Struts y por supuesto la invocación de funciones para el refresco de información en tiempo real mediante **AJAX** y, más concretamente la librería **Direct Web Remoting** explicada en el siguiente punto.

Por último decir que también se utilizan hojas de estilo **CSS (Style Sheets)** para uniformar la presentación de los datos al usuario a través de todas las páginas que componen la aplicación.

4.6.2 AJAX Y DWR (DIRECT WEB REMOTING)

AJAX, acrónimo de **Asynchronous JavaScript And XML** (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

En la Aplicación Web de Supervisión, se ha hecho uso de una librería Java que permite interaccionar al servidor Java con el navegador del cliente que tiene código escrito en javascript.

Esta librería se llama **Direct Web Remoting (DWR)**. Su manejo es fácil y muy versátil para el desarrollador de la aplicación web. Evita la utilización del siempre engorroso objeto XMLHttpRequest y emplea AJAX de forma completamente transparente al desarrollador.

Básicamente lo que tiene que hacer el desarrollador es declarar una serie de objetos para que se permita la conversión entre el objeto Java y el objeto javascript, así como definir en un fichero el mapeo de distintas funciones javascript a métodos Java que se ejecutan en el Servidor.

El funcionamiento, por tanto, es asíncrono de modo que al código javascript se le dice cual será el *callback* o función a ejecutar cuando el Servidor devuelva sus resultados. Mientras tanto el código javascript seguirá ejecutando sus operaciones.

A la vuelta del resultado, se realizará automáticamente la conversión de los objetos Java a javascript y quedan listos para ser presentados al usuario.

Se trata, por tanto, de una tecnología bajo la cual subyace AJAX y que resulta fácil de configurar y de manejar por parte del desarrollador de la aplicación al que además ofrece un amplio abanico de posibilidades de utilización.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en la Referencias se indica la URL de la página web oficial de DWR.

4.6.3 API DE GOOGLE MAPS

Google Maps nos ofrece mapas desplazables del mundo entero, fotos satelitales, la ruta más corta entre diferentes ubicaciones y muchas otras características accesibles por medio de ciertos procedimientos. Google Maps es fácilmente integrable en cualquier sitio web. Para ello, Google pone a disposición de los desarrolladores el API de Google Maps.

Mediante este API, Google nos proporciona una serie de procedimientos bien documentados para que, a través de código JavaScript, nuestra página web pueda comunicarse de alguna manera con sus servidores solicitándoles cierta información sobre los mapas o realizar ciertas actuaciones sobre los mismos. Para poder utilizar este API, todo lo que debemos hacer es solicitar nuestra API Key e indicar en qué URL vamos a alojar nuestras aplicaciones

Concretamente en la Aplicación Web de Supervisión, el API de Google Maps ha sido empleado para desarrollar la funcionalidad correspondiente a la Monitorización del Centro de Control donde se ofrece al usuario un mapa completo de la autopista, con marcas en cada una de las estaciones.

Se ha hecho una integración de la funcionalidad proporcionada por el API de Google Maps con el motor **DWR** de **AJAX** para mostrar sobre el propio mapa información relevante de cada estación en tiempo real, además de señalar también en tiempo real las estaciones que tiene algún tipo de alarma y otra funcionalidad que será descrita con más detalle tanto en el apartado dedicado al Desarrollo de la Aplicación como en el apartado dedicado a la Descripción Funcional de la aplicación.

4.6.4 YAHOO USER INTERFACE

Se trata de una serie de bibliotecas escritas en **JavaScript**, que proporcionan un conjunto de utilidades y controles para desarrollar aplicaciones web interactivas usando técnicas DOM, DHTML y AJAX. **YUI** incluye también un conjunto de fuentes **CSS**. Todos los componentes **YUI** han sido desarrollados en Open Source bajo BSD (Berkeley Software Distribution) License y está disponible para todo tipo de uso.

Yahoo User Interface Library tiene un conjunto de componentes que están agrupados en Utilities y Controls.

El *"YUI Controls"* brinda elementos de diseño altamente interactivos para las páginas web. Algunos de estos controles se describen a continuación:

- **AutoComplete**: este control permite al usuario interacción en formulario de ingreso brindando listas de sugerencia
- **Calendar**: este control un calendario grafico con selección dinámica de fechas.
- **Container**: este control soporte diferentes patrones DHTML con ToolTip, Panel, Dialog.
- **Menu**: aplicación para crear menús en el instante con pocas líneas de código.

Todos estos componentes vienen con una documentación muy detallada y con muy buenos ejemplos.

En la aplicación web concretamente se ha hecho uso de la amplia gama de posibilidades que brinda **YUI** para construir tablas dinámicamente con javascript bajo la tecnología **AJAX**. Ofrece una serie de constructores con un gran abanico de posibilidades tanto para construir las tablas a medida que se va refrescando el conjunto de resultados en tiempo real como para decorar esas tablas mediante las hojas de estilo (**CSS**) disponibles a tal efecto.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la aplicación. Por otra parte en la Referenciasse indica la URL de la página web oficial de YUI.

4.6.5 DISPLAY TAG LIBRARY

La **"Display Tag Library"** es una librería open source que trabaja en un modelo MVC e incluye un amplio abanico de etiquetas de gran utilidad cuando se quiere mostrar una colección de datos en un tabla. Además permite, entre otras cosas, agregarle estilos, decoradores a los datos mostrados, exportar a distintos formatos, utilizar paginación y *"sort"* tanto por medio de la librería como de forma externa.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en la Referencias se indica la URL de la página web oficial de Display TagLibrary.

4.6.6 SPRING

Spring es un conjunto de librerías "a la carta" de entre las que podemos escoger aquellas que faciliten el desarrollo de nuestra aplicación. Entre sus posibilidades más potentes está su contenedor de Inversión de Control, también llamado Inyección de Dependencias que es una técnica alternativa a las clásicas búsquedas de recursos vía JNDI. Permite configurar las clases en un archivo XML y definir en él las dependencias. De esta forma la aplicación se vuelve muy modular y a la vez no adquiere dependencias con Spring, la introducción de aspectos, plantillas de utilidades para Hibernate, iBatis y JDBC.

Ayuda a que los diferentes componentes que forman una aplicación trabajen entre sí, pero no establece apenas dependencias consigo mismo. Esta es la primera característica de este framework. Sería posible retirarlo sin prácticamente cambiar líneas de código. Lo único que sería necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro framework similar o mediante nuestro código. De hecho, Spring soporta una serie de componentes y clases que son capaces de gestionar también la capa de presentación, por lo que podríamos sustituir la capa de JSF por una compuesta por Spring MVC, que es realmente como se denomina a esta capa.

Además de todo esto, Spring soporta una serie de librerías y utilidades que dan una facilidad añadida a la recepción y tratamiento de la información desde Web Services. Esto es importante a la hora de tener en cuenta que muchas de las comunicaciones con sistemas externos a la propia aplicación van a hacerse por medio de este tipo de interfaces.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en la Referencias se indica la URL de la página web oficial de Spring.

4.6.7 APACHE STRUTS

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en las que Java Entreprise esté disponible lo convierten en una herramienta altamente disponible.

Con Struts resulta muy fácil manejar el intercambio de datos entre las capas de presentación y de negocio, así como definir el flujo de acciones que gestiona dicho intercambio además de establecer reglas de validación de formularios. Será el servlet controlador proporcionado por el API de Struts el encargado de manejar este flujo de datos y acciones.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en la Referencias se indica la URL de la página web oficial de Struts.

4.6.8 ACEGI

Como se ha dicho anteriormente, la aplicación web de Supervisión pretende aunar en una aplicación todos los servicios y funcionalidades a las que hasta ahora se accedía por medio de distintas aplicaciones. Esto implica que a la misma aplicación accederán personas con muy diversos roles y capacidades de acceso. Accederán peajistas, controladores, supervisores, personal de mantenimiento, etc…

La seguridad y el control de acceso a las distintas partes de la aplicación son, por tanto, elementos fundamentales a tener en cuenta tanto en la fase de análisis como en la fase de desarrollo de la aplicación. Para este cometido se ha escogido el framework **Acegi Security**.

ACEGI es un framework de seguridad opensource que permite mantener a la lógica de negocio libre de código de seguridad.

Proporciona 3 opciones básicas de seguridad:

- Listas de control de acceso (ACL) web basadas en esquemas URL
- Protección de métodos y clases Java usando AOP
- Single sign-on (SSO)

Seguridad proporcionada por el contenedor Web

ACEGI es, en definitiva, un gestor de seguridad diseñado para ser utilizado con Spring puesto que comparte con él mecanismos de configuración. Proporciona una capa que envuelve diversos estándares de seguridad presentes en Java y ofrece una forma unificada de configuración a través de un descriptor en XML.

Cubre la capa web y la de negocio. A nivel Web captura todas las peticiones mediante la implementación de un filtro y a nivel de métodos mediante interceptación a través de AOP. En ambos casos permite aplicar los criterios de seguridad que trae de serie o añadir nuevas opciones de forma sencilla implementando los interfaces diseñados a tal fin.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en las Referencias se indica la URL de la página web oficial de Acegi.

4.6.9 HIBERNATE (CAPA DE PERSISTENCIA)

La capa de persistencia debe permitir mapear un modelo de clases definido a un modelo relacional, sin que se tengan restricciones en ambos diseños. Por ello se ha elegido **Hibernate**.

Además es un framework que se encuentra en continua evolución, por lo que está garantizada su escalabilidad.

En la parte dedicada al Desarrollo de la Aplicación se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en las Referencias se indica la URL de la página web oficial de Hibernate.

4.6.10 JAXB (JAVA ARCHITECTURE FOR XML BINDING)

JAXB proporciona un puente entre la tecnología de representación de datos **XML (Extensible Markup Language)** y la tecnología Java.

JAXB incluye un compilador que asocia un esquema a un conjunto de clases Java. Una vez que tengamos nuestras clases, podremos construir las representaciones de objetos Java de los datos XML que siguen las reglas que el esquema define.

Al igual que un documento XML es un ejemplar de un esquema, un objeto Java es un ejemplar de una clase. Así, JAXB permite que creemos los objetos Java en el mismo nivel conceptual que los datos XML. La representación de nuestros datos de esta manera permite que los manipulemos de manera semejante como manipularíamos objetos de Java, haciendo más fácil la creación de aplicaciones para procesar datos XML. Una vez que tengamos nuestros datos en la forma de objetos Java, es fácil acceder a ellos.

Además, después de trabajar con los datos, podemos escribir los objetos Java en un nuevo documento XML. Con el acceso fácil a los datos XML que proporciona JAXB, solamente necesitamos escribir aplicaciones que realmente utilizarán los datos, en vez gastar el tiempo en escribir código para formatear los datos.

En nuestra Aplicación Web hemos hecho un mapeo del esquema MensajeWeb.xsd descrito anteriormente en objetos Java. De esta manera desciframos el contenido encapsulado en los mensajes de información que le proporciona el Servidor en Tiempo Real a la aplicación web y lo almacenamos en objetos Java que posteriormemente se envían al cliente. El motor de la librería DWR será el encargado de realizar la posterior transformación en objeto javascript de modo que pueda ser manejado y representado por funciones javascript por el navegador, así como los datos tabulados utilizando los constructores que proporción YUI escritos en javascript.

4.6.11 APACHE LOG4J

LOG4J es una herramienta que permite centralizar y administrar las trazas y mensajes de debugging y avisos de la aplicación de manera fácil y altamente configurable.

Permite habilitar y deshabilitar ciertos logs, mientras otros no sufren ninguna alteración. Esto se realiza categorizando los mensajes de logs de acuerdo al criterio del programador.

Proporciona los siguientes niveles de prioridad:

- **FATAL**: se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje el programa abortará.
- **ERROR**: se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **WARN**: se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan al correcto funcionamiento del programa.
- **INFO**: se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.
- **DEBUG**: se utiliza para escribir mensajes de depuración. Este nivel no debe estar activado cuando la aplicación se encuentre en producción.
- **TRACE**: se utiliza para mostrar mensajes con un mayor nivel de detalle que debug

La configuración de las trazas de la aplicación viene definida en el fichero **log4j.xml.** En este fichero XML se ha especificado que las trazas generadas por Hibernate, Spring y las de la propia aplicación se vuelquen en ficheros independientes cada uno con su nivel de prioridad.

En la parte dedicada al Desarrollo del Software se describe con detalle la configuración y el uso de esta tecnología en la Aplicación. Por otra parte en la Referencias se indica la URL de la página web oficial de Apache Log4j.

5 HERRAMIENTAS Y ENTORNOS DE TRABAJO

5.1.ENTORNO DE DESARROLLO (IDE)

Como entorno de desarrollo, se ha utilizado **Eclipse**. Se trata de un IDE sólido, con soporte para muchas de las características avanzadas de edición en Java y con un ritmo de desarrollo muy rápido.

Por otra parte existe una gran cantidad de plug-ins que le da una cobertura mayor a todas las necesidades que se pueden plantear a la hora de un desarrollo Web por muy compleja que sea la tecnología o los frameworks empleados. Además se trata de un IDE gratuito.

5.2.SERVIDOR DE APLICACIONES

Como servidor de aplicaciones se utiliza uno opensource que permite soportar toda la arquitectura planteada, en concreto nos referimos a **Tomcat**.

Tomcat es un Servidor de Aplicaciones de código abierto basado en Java, y esta es precisamente su mayor ventaja, ya que permite que pueda "correr" sobre cualquier sistema operativo que soporte una máquina virtual de Java. Implementa todo el paquete de servicios del estándar J2EE, por lo que soporta sin problemas todas las especificaciones definidas para la arquitectura propuesta.

5.3.BASE DE DATOS

El Sistema de Gestión de Bases de Datos Relacional propuesto es **SQL Server 2000**, puesto que se trata de un sistema altamente fiable. Es un sistema que contiene un lenguaje SQL estándar y normalizado, por lo que la capa de persistencia puede ser implementada sin mayor problema. De todas formas, la propia

filosofía de la arquitectura propuesta permite la utilización de cualquier tipo de motor de bases de datos propuesto, ya que los cambios serían simples en los mapeos de persistencia.

6 ARQUITECTURA DEL SISTEMA DE TRANSFERENCIA DE MENSAJES

6.1. DESCRIPCIÓN DE LA ARQUITECTURA DE COMUNICACIONES

El funcionamiento del sistema de supervisión y monitorización de peajes se fundamenta en una arquitectura de comunicaciones que posibilita la centralización en tiempo real de la información proveniente de todas las entidades involucradas en el sistema.

La Figura 2 muestra un esquema que describe gráficamente la arquitectura de comunicaciones diseñada.

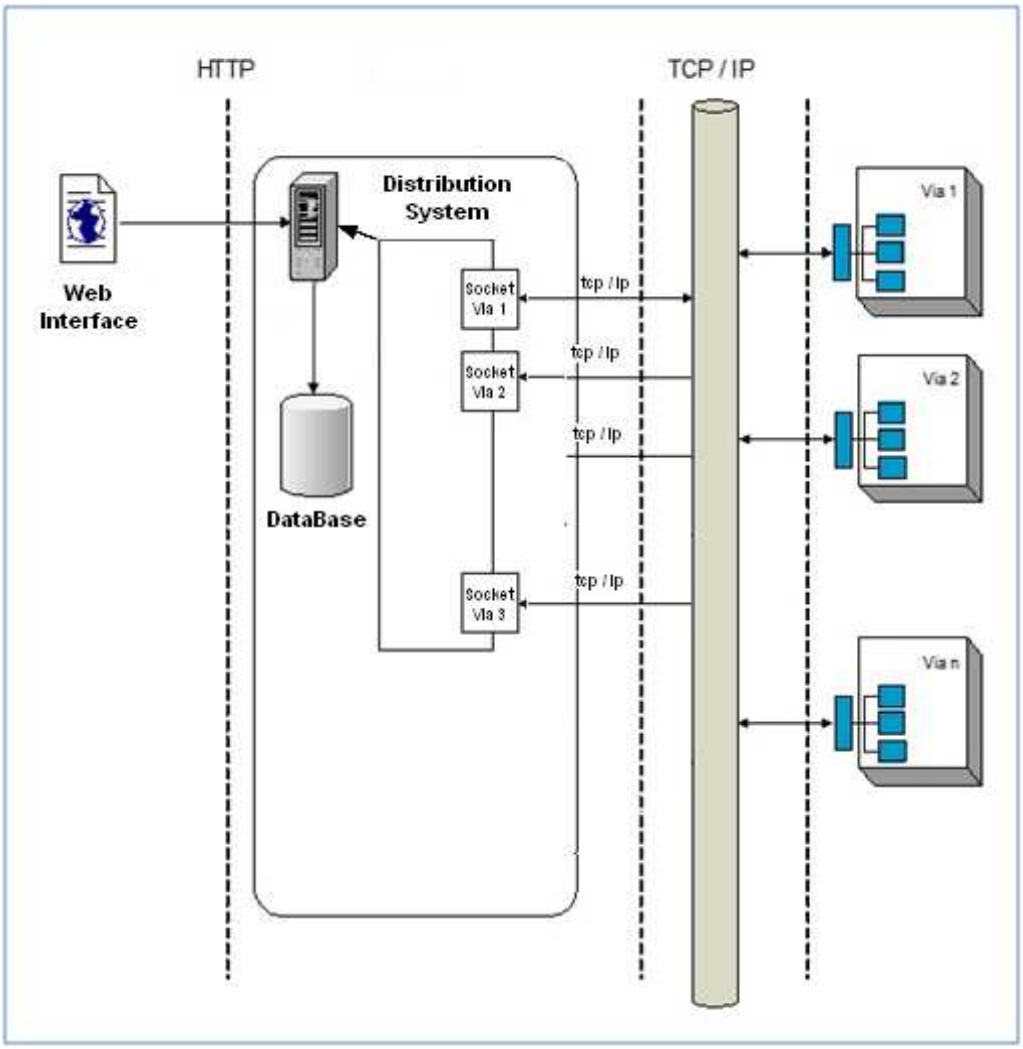

Figura 3: Arquitectura de comunicaciones

Las vías se encuentran en el nivel inferior de la arquitectura. Empleando el protocolo TCP/IP existirá una comunicación entre las vías y el Centro de Control, de modo que este tenga en todo momento información actualizada del estado de todos los componentes y dispositivos del peaje, así como de posibles alarmas generadas.

Será el protocolo de transporte TCP el encargado de gestionar la posible pérdida de paquetes derivada de eventuales caídas en las comunicaciones o errores de conexión entre los distintos agentes involucrados en la transferencia de los mensajes.

Una vez almacenada toda esta información en el proceso encargado de mapear el peaje a nivel de Centro de Control, será el protocolo HTTP el encargado de transferir la información entre este nivel y la Aplicación web de Supervisión.

La Aplicación Web de Supervisión se encuentra en el nivel más alto de la arquitectura, mostrando al usuario información en tiempo real bajo demanda de cualquier emplazamiento o dispositivo del peaje.

Además el sistema almacenará información relevante en sus propias Bases de Datos para su posterior consulta a petición del usuario. La Base de Datos empleada por la aplicación será única e independiente del resto del sistema.

En el apartado siguiente se explica con más detalle el funcionamiento de este sistema de transferencia de mensajes, así como los distintos componentes involucrados y las distintas funciones que desempeñan en este esquema.

6.2. DESCRIPCIÓN DE LOS COMPONENTES INVOLUCRADOS

En la Figura 4 se muestra un esquema donde aparecen todos los elementos involucrados en el sistema de transferencia de mensajes. En primer lugar se describe el funcionamiento general del sistema así como la interrelación y la función de cada uno de los agentes implicados. Tras esta visión general del sistema se pasa a enumerar y a hacer una breve descripción de cada uno de ellos así como indicar la función que cumplen y cómo se han implementado.

Como se ha explicado en el apartado anterior, las vías se encuentran en el nivel inferior, las cuales tienen sus propias bases de datos que almacenan la información necesaria para que los procesos que la controlan funcionen correctamente. En cada una de las vías se instala una entidad de comunicaciones llamada **"Servidor de Alarmas"**. El Servidor de Alarmas está escuchando en un puerto configurable de manera que cuando el **"Cliente de Alarmas"** instalado en la entidad jerárquicamente superior se conecte a él, volcará toda la información que tenga acerca de esa vía.

El Servidor de Alarmas se nutre de una **.DLL (Data Link Library)** instalada en el proceso controlador de vía que le proporciona información acerca de los cambios de estado en los elementos, modo de funcionamiento o alarmas que se produzcan. Se trata de una librería desarrollada a tal efecto integrada en el software de la vía, cuyo cometido es generar los mensajes XML de información del sistema de peaje a partir de la ristra de bytes que genera el software controlador de vía. Una vez generado el mensaje, cuya estructura obedece al esquema de información de vía, la librería se encarga de escribirlo en una cola de mensajes, donde queda almacenado en espera de que la entidad de nivel superior proceda a su extracción y retransmisión.

El esquema de información de vía se ha diseñado de manera que la transferencia de mensajes sea lo más eficiente posible así como que sea capaz de cubrir posibles necesidades futuras. Este esquema es analizado en un apartado posterior. De esta manera, las diferentes entidades involucradas en la transferencia de los mensajes XML desde el nivel de vía hasta el nivel de centro no realizan ninguna labor de parseo ni interpretación de los mensajes enviados, sino que tan sólo se encargan de transferir los mensajes de información de las vías a la entidad superior tal y como se lo entregó la entidad inferior.

El proceso controlador de vía tan sólo enviará mensajes de información, estados y alarmas al Servidor de Alarmas cuando se produzca un cambio de estado, y no continuamente, con el fin de no saturar las comunicaciones en el nivel superior donde la avalancha de mensajes procedente de todas las vías del peaje es muy grande.

Como se puede observar en el esquema mostrado, existen Clientes y Servidores de Alarmas en cada uno de los niveles excepto en el nivel inferior en el cual sólo hay un Servidor de Alarmas. Cada nivel se conectará como cliente al puerto configurado en el servidor del nivel inferior, momento en el cual las alarmas extraídas son almacenadas en una cola local. Cuando el Cliente de Alarmas de nivel superior se conecta al Servidor de Alarmas, este extraerá los mensajes almacenados en dicha cola y se los enviará tal y como se recibieron del nivel inferior. De esta forma, los mensajes fluyen de unos niveles a otros hasta llegar al Centro de Control.

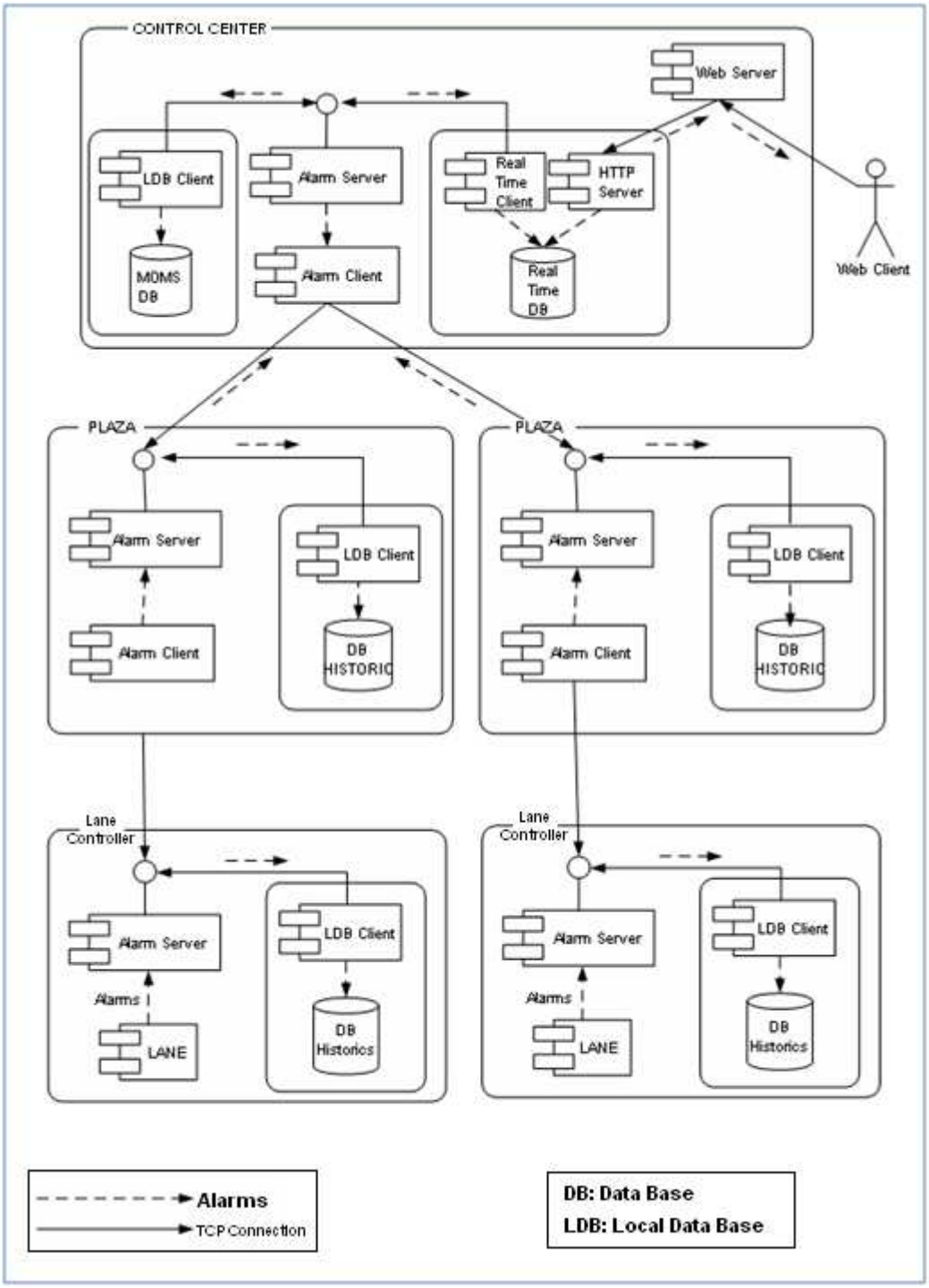

Figura 4: Arquitectura de transferencia de mensajes

En el nivel superior, donde se aloja la aplicación de Supervisión, reside el "Servidor en Tiempo Real". Se trata de un módulo de almacenamiento que mantiene en memoria un mapa completo de todo el peaje con la información en tiempo real que le llega desde las vías. Cuando el usuario de la aplicación web de Supervisión solicita cierta información, la aplicación abrirá una conexión HTTP con el Servidor

en Tiempo Real para extraer la información requerida. De esta forma, el usuario de nivel superior siempre tendrá disponible la información del peaje en tiempo real y esta será accesible desde un punto del sistema.

El esquema de los mensajes XML intercambiados entre el Servidor en Tiempo Real y la aplicación web tiene un esquema diferente que llamaremos esquema XML Web para distinguirlo del esquema intercambiado por el resto de agentes de la arquitectura. Este esquema es más apropiado para el manejo por parte de la aplicación web de la información que se debe mostrar al usuario. De este modo, el Servidor en Tiempo Real realiza a su vez una labor de parseo y traducción entre ambos esquemas de mensaje. Ambos esquemas son explicados en detalle posteriormente.

Tanto las vías como las estaciones como el Centro de Control tienen sus propias Bases de Datos locales que almacenan la información de configuración y operación necesaria para que tales entidades funcionen correctamente.

La aplicación de Supervisión tiene su propia Base de Datos en la que se guarda información de configuración de la aplicación, así como históricos de los distintos eventos que se van produciendo y alarmas que se van generando en el sistema, de modo que esta información quede almacenada de manera estable para ulteriores consultas, análisis de fallos del sistema y posibles auditorías. Para esta labor existen procesos que se encargan de almacenar esta información en los históricos de la base de datos al mismo tiempo que se alimenta el Servidor en Tiempo Real de información en tiempo real.

A continuación se enumeran y describen brevemente los procesos y entidades que componen la arquitectura descrita.

6.2.1. PROCESO CONTROLADOR DE VÍA

Es el software que corre en cada una de las vías del peaje. Este software, con ayuda de varios procesos corriendo al mismo nivel, se encarga de controlar absolutamente todo lo que ocurre al nivel de vía. Controla todos los dispositivos de la vía como pueden ser barreras, semáforos, antenas, pisones, cortinas, display, etc…

El software de la vía permite abrir turnos en distintos modos, ya sea manual, automático o modo telepeaje. El proceso controlador de vía deberá aplicar la tarifa correcta a los clientes que tiene almacenada en su base de datos local en función de la categoría de vehículo detectada por la cortina.

El proceso controlador de vía envía ristras de bytes con diferentes formatos. Envía mensajes periódicos emitiendo cierta información acerca de su configuración. Además envía mensajes cuando existen cambios de estado en sus dispositivos o se generan alarmas en alguna de ellos.

Al proceso controlador de vía se le ha añadido una librería Data Link Library de entrada/salida que tiene como misión parsear e interpretar las ristras de bytes que genera la vía y almacenarlos una vez traducidos según el esquema XML definido en una cola (queue) de mensajes configurada al efecto y que está en la misma máquina donde corre el proceso controlador de vía.

En esa cola quedarán los mensajes almacenados en espera de que alguien los extraiga.

6.2.2. SERVIDOR DE ALARMAS

El módulo Servidor de Alarmas tiene un canal de entrada que es la cola de mensajes donde se almacenan los mensajes generados por la vía y un canal de salida mediante sockets TCP que no es más que un socket servidor escuchando en un puerto configurable.

Su funcionamiento es muy simple. Cuando algún cliente se conecta al servidor en el que está escuchando, este le replica a dicho cliente todos los mensajes que tenga almacenados en la cola de donde los lee. Será cometido del cliente, en su caso, descartar los mensajes que no vayan dirigidos hacia él.

En ningún caso el Servidor de Alarmas interpreta los mensajes XML. Se limita a enviar los mensajes a los clientes que se conectan tal y como otro proceso los escribió en la cola de mensajes.

Con estos dos elementos ya descritos, es fácil imaginarse cual será el esquema a nivel de vía. La Figura 5 lo muestra gráficamente.

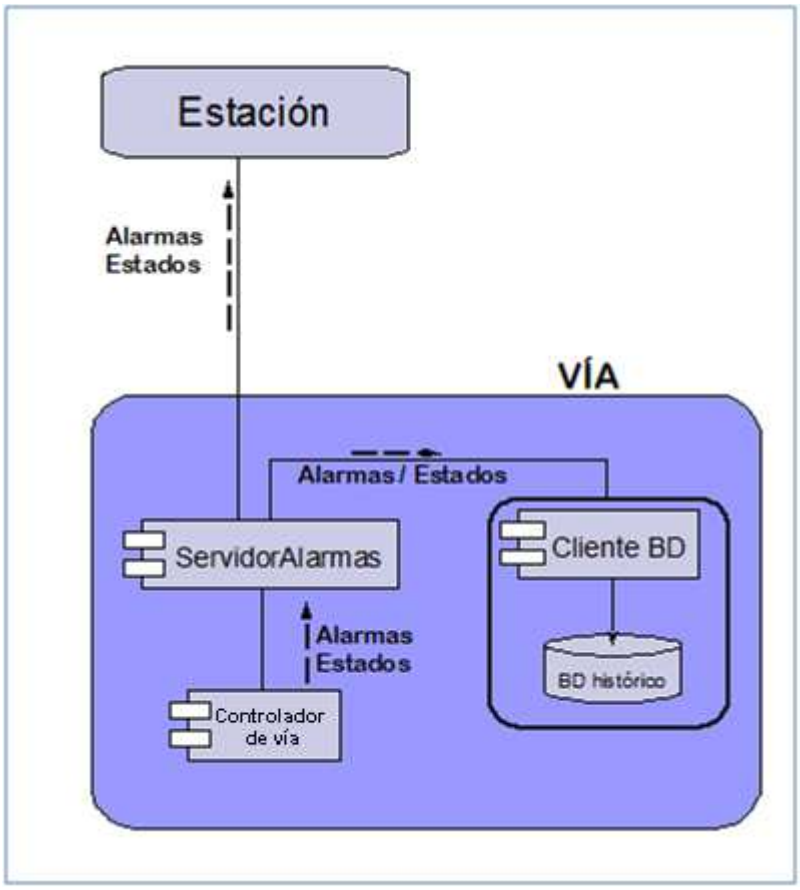

Figura 5: Agente de transferencia a nivel de vía

6.2.3. CLIENTE DE ALARMAS

El Cliente de Alarmas es el módulo opuesto al Servidor de Alarmas. Si este escucha en un puerto y lee de una cola, aquel se conecta como cliente a un puerto y escribe en una cola.

De esta forma, el Cliente de Alarmas de nivel superior, se encargará de conectarse como cliente al Servidor de Alarmas que escucha en el nivel inferior, momento en el cual recibirá de este los mensajes almacenados en su cola y se encargará de almacenarlos en la cola configurada en su propio nivel.

Es así como conseguimos mediante conexiones TCP/IP cliente-servidor la transferencia del mensaje XML entre dos agentes a distinto nivel jerárquico.

Además, el Cliente de Alarmas se encarga de generar una alarma cuando se pierde o se recupera la comunicación con el servidor de alarmas con el que está conectado.

La Figura 6 muestra un esquema que describe gráficamente lo recién explicado.

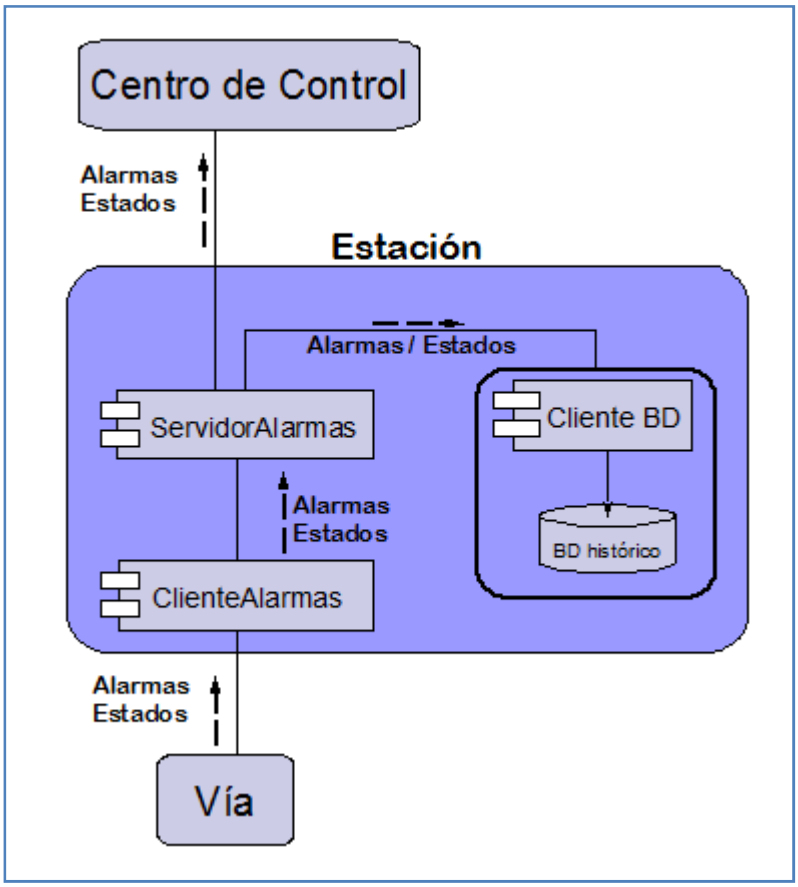

Figura 6: Agente de transferencia a nivel de estación

6.2.4. SERVIDOR EN TIEMPO REAL

El Servidor en Tiempo Real es un proceso desarrollado en lenguaje Java encargado de mantener en todo momento un mapa de memoria del peaje con el objetivo de proporcionar a la aplicación web de Supervisión información en tiempo real bajo demanda de cualquier localización del peaje a través del protocolo HTTP en un puerto configurado a tal efecto. Se trata, pues, del último nivel a donde se transfiere la información del sistema antes de ser proporcionada a la aplicación web encargada de servir de interfaz al usuario y de mostrarle dicha información.

El nombre de Servidor en Tiempo Real viene precisamente de esta aspiración de mantener una "memoria" de acceso rápido que permita a la aplicación web mostrar en tiempo real y siempre que el usuario lo solicite la información demandada.

El Servidor en Tiempo Real recibe la información del sistema de peaje según la estructura de XML del sistema de peaje, la parsea y almacena en el mapa del peaje que mantiene en memoria encapsulado en un mensaje con esquema XML_Web para proporcionar a la Aplicación web la información solicitada con el menor retardo posible.

La Figura 7 representa esquemáticamente este comportamiento y la función del Servidor en Tiempo Real representado aquí como WebService.

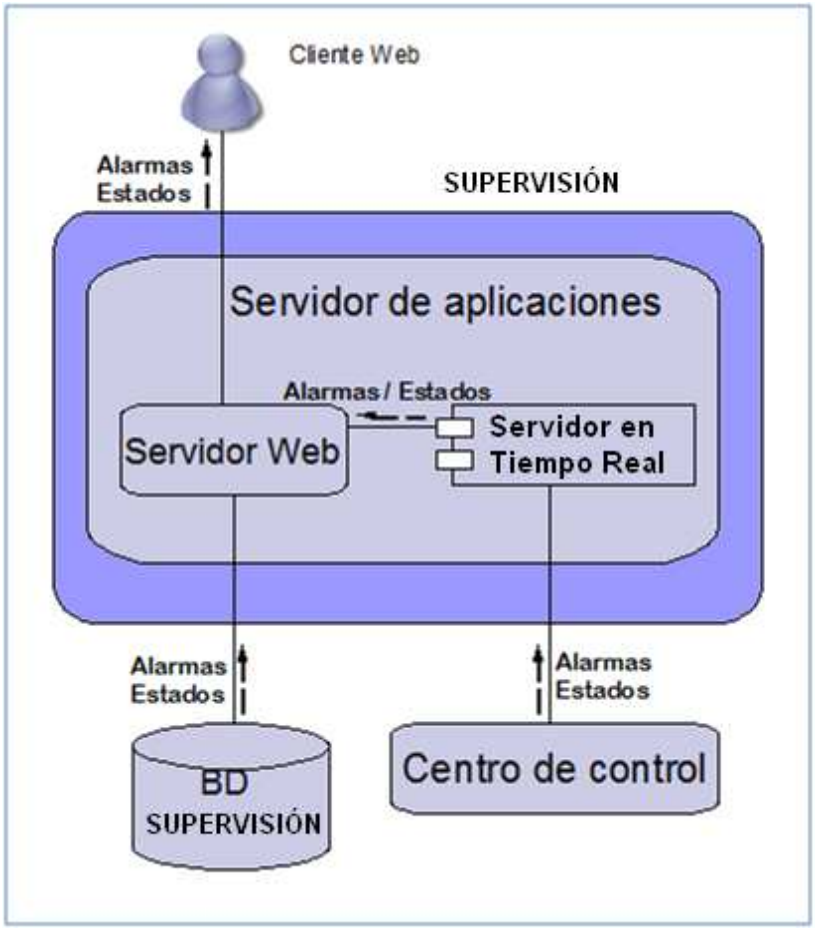

Figura 7: Agente de transferencia a nivel de Centro de Control

6.2.5. BASE DE DATOS DE SUPERVISIÓN

La aplicación web de Supervisión emplea su propia Base de Datos donde se guarda toda la configuración necesaria para el correcto funcionamiento de la aplicación, así como tablas de históricos en las que se van insertando registros de estados y alarmas del sistema a medida que esta información va llegando a este nivel.

El modelo de Datos diseñado para este sistema se explica con detalle en el apartado dedicado al Modelo de Datos.

6.2.6. SERVIDOR WEB

El Servidor Web será el encargado de alojar la aplicación web y de mostrar la información al usuario a través de peticiones http. Las características técnicas de este servidor web se describen en el apartado dedicado a la Descripción Técnica del Sistema.

El servidor web tendrá un doble acceso:

- Por un lado el acceso a las Bases de Datos que contiene información necesaria para el correcto funcionamiento del sistema así como información de configuración e históricos.
- Por otro lado, accederá al Servidor en Tiempo Real explicado en el punto anterior para mostrar al usuario toda aquella información que se requiere mostrar en tiempo real.

Tras extraer la información requerida por el usuario, debe devolverle a este por medio del protocolo HTTP la página HTML correctamente compuesta y maquetada de modo que le permita al navegador mostrar la información solicitada.

6.2.7. CLIENTE WEB

Será un navegador web que se encargará de mostrar al cliente la representación gráfica de la información obtenida del Servidor en formato HTML, así como de servir de interfaz con el usuario para permitir a este hacer uso de la funcionalidad proporcionada por la aplicación.

6.3.MENSAJES DE INFORMACIÓN DEL SISTEMA DE PEAJE

En el apartado anterior se ha explicado el diseño y el funcionamiento de la arquitectura del sistema de comunicaciones que permite la transferencia de mensajes desde las entidades de nivel inferior que generan la información hasta el nivel superior que la almacena y se la proporciona a la aplicación web.

En este apartado se va a explicar la estructura de los mensajes intercambiados entre los diferentes agentes y la forma en que la información se encapsula para ser transportada a través de la red hasta su destino final.

Como ya se ha dicho anteriormente, la estructura del mensaje que maneja el sistema de peaje es distinta de la que maneja la aplicación web. De esta manera, el módulo de almacenamiento llamado Servidor en Tiempo Real hace la doble labor de almacenar la información del mapa del peaje, y de parsear el mensaje atendiendo a la estructura utilizada por el sistema de peaje y traducirla al esquema utilizado por la aplicación web mucho más adecuado para el tratamiento a este nivel.

Ambas estructuras de mensaje XML están definidas según unos esquemas XSD que llamaremos **Mensaje.xsd** y **MensajeWeb.xsd** para distinguirlos. Ambos son explicados a continuación.

6.3.1. MENSAJE.XSD

El sistema está jerarquizado y toda la información se agrupa en el Centro de Control del sistema.

La estructura del mensaje XML está pensada para que se pueda agrupar la información proveniente de las vías del sistema de peaje.

Cuando en una vía o elemento del peaje se produce un evento, la información de este suceso se encapsula en un mensaje XML y se retransmite a través del sistema de distribución hacia el Centro de Control, para que este lo registre. Este tipo de información fluye desde las capas inferiores del peaje hasta la superior y lo hace de forma espontánea.

6.3.1.1 VISIÓN GLOBAL

La estructura del mensaje XML está pensada para que se pueda agrupar la información proveniente de las vías del sistema de peaje.

Los mensajes XML no son generados continuamente ni de forma periódica por parte de las vías, sino que se generan cada vez que existe un cambio de estado en algún elemento o se produce alguna alarma en algún dispositivo. Esto evita en gran medida la congestión de las redes de comunicaciones cuando hay muchas vías generando mensajes de información al mismo tiempo.

Si en una vía se produce un cambio de estado y una alarma en un elemento a la vez, la vía puede empaquetar toda la información en un solo mensaje y transmitirlo; pero esto no es obligatorio, de hecho el funcionamiento tenderá siempre a empaquetar eventos individuales en mensajes independientes.

Los mensajes estarán etiquetados con un origen y un destino. Todo origen o destino será un **"ID_LUGAR"** [1] en forma de cadena de texto y representando el número en formato hexadecimal.

Esta información tendrá como destinatario cualquiera que esté conectado a través del sistema de conexión, es decir se enviará con el campo DESTINO con valor 0xFFFFFFFF (dirección de broadcast). De esta manera todo aquél sistema que se encuentre conectado recibirá la información. De él depende qué hacer con toda la información que pueda recibir.

El elemento situado en el Centro de Control funcionará en "modo promiscuo" para capturar todos los mensajes y registrar todo lo que ocurre en el sistema de peaje. También se almacenará cierta información en la base de datos cuando se quiera hacer una posterior consulta o tratamiento de históricos.

[1] Identificador único de la ubicación en el sistema de peaje. Mirar tabla LUGARES en el apartado Modelo de Datos.

6.3.1.2 DESCRIPCIÓN DE LOS TAGS

La Figura 8 representa el esquema XSD que rige la estructura que deben tener los mensajes de información que se transmiten hacia el Centro de Control por medio del sistema de distribución de mensajes.

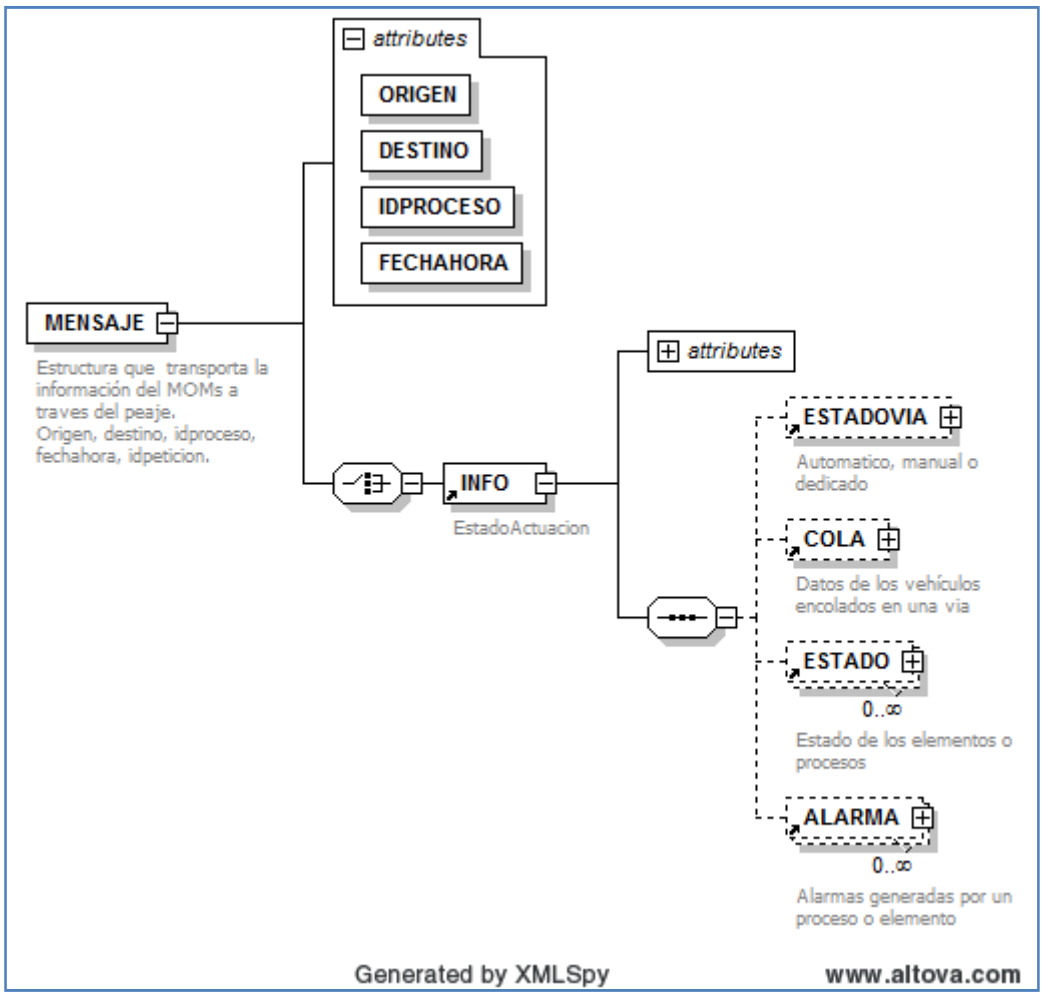

Figura 8: Mensaje de distribución de vía

A continuación pasamos a describir una a una las etiquetas que componen este mensaje.

ETIQUETA <MENSAJE>

Esta es la etiqueta principal. Engloba todo el mensaje. Sólo puede haber una por mensaje y ha de llevar los siguientes atributos:

- **ORIGEN**: cadena de texto con el ID_LUGAR que genera el mensaje.
- **DESTINO**: cadena de texto con el ID_LUGAR de los destinatarios.
- **IDPROCESO**: entero que indica el proceso que envía la información. Los códigos que identifican los procesos son los de la tabla DESC_PROCESO descrita en el apartado referente al Modelo de Datos.
- **FECHAHORA**. fecha y hora en la que se genera el mensaje.

Además la etiqueta MENSAJE contendrá la etiqueta **<INFO>**. Esta etiqueta encapsula el envío de información que se transmite a través del sistema, desde un punto del sistema a otro.

ETIQUETA <INFO>

Etiqueta que encapsula el envío de información a través del sistema. La información va de punto a punto es decir, no puede enviarse a más de un lugar.

ETIQUETA <ESTADOVIA>

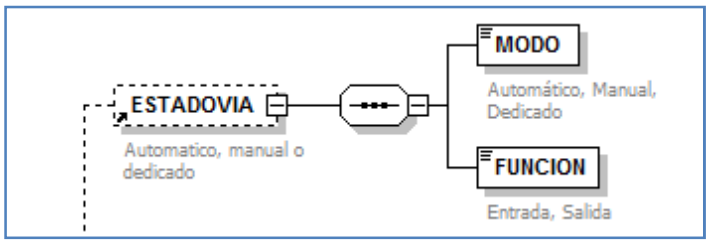

Figura 9: Etiqueta ESTADOVIA

- **<MODO>** Entero que describe el modo en que funciona la vía. Este código se corresponde con alguno de los identificadores configurados en la tabla DESC_MODO_VIA descrita en el apartado referente al Modelo de Datos.

Según este mismo apartado, los posibles modos en que opera la vía son:

1. Manual: hay un peajista físicamente en la vía
2. Automático: una máquina de cobro automática o expendedora de tickets
3. Telepeaje: lectura automática de tarjetas

- **<FUNCION>** Entero que indica si la vía está abierta como vía de entrada o como vía de salida. Este código se corresponde con alguno de los identificadores configurados en la tabla DESC_FUNCION_VIA descrita en el apartado referente al Modelo de Datos.

1. Entrada: funciona como tiquetera.
2. Salida: salida del peaje, cobra.

ETIQUETA <COLA>

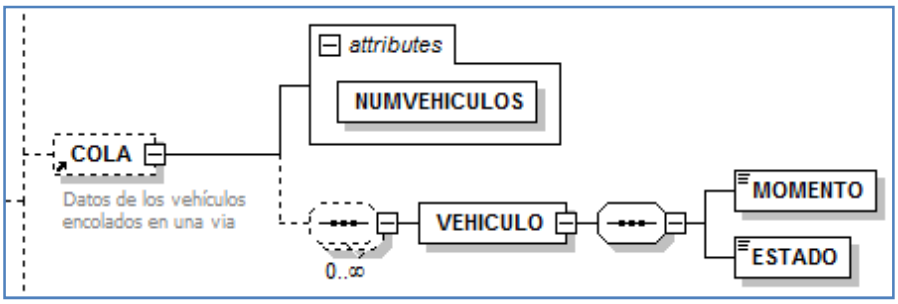

Figura 10: Etiqueta COLA

Nos da la cola de vehículos que tiene una vía.

Lleva un atributo obligatorio que es **NUMVEHICULOS** que indica el número de vehículos en la cola. Esta etiqueta contendrá una secuencia de etiquetas **VEHICULO**, tantas como NUMVEHICULOS indique.

- **<VEHICULO>** Contiene datos de un vehículo en una cola. Esta lista de vehículos da el orden de los vehículos en la vía, es decir el primer vehículo será el primero en esta secuencia.
- **<MOMENTO>** Entero que indica la posición del vehículo en el sistema de clasificación.
- **<ESTADO>** Entero que indica el estado de ese vehículo.

ETIQUETA <ESTADO>

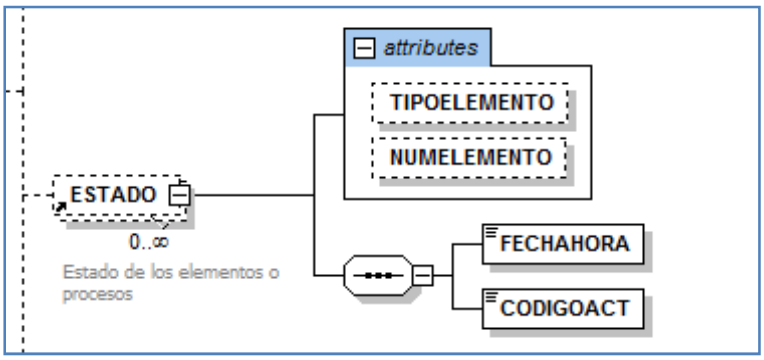

Figura 11: Etiqueta ESTADO

Etiqueta que se puede repetir tantas veces como sea necesario, que agrupa la información del estado de un elemento concreto que identifica con 2 atributos obligatorios:

- **TIPOELEMENTO:** atributo de valor entero que indica el tipo de elemento. Este campo se corresponde con alguno de los elementos dados de alta en la configuración de la tabla DESC_ELEMENTO descrito en el apartado referente al Modelo de Datos.
- **NUMELEMENTO** atributo de valor entero que indica el número de elemento. Se refiere al identificador del elemento entre varios del mismo tipo (con el mismo TIPOELEMENTO).

La información sobre el estado del elemento se representa con las etiquetas:

- **<FECHAHORA>:** fecha y hora asociadas al estado del elemento.
- **<CODIGOACT>**: entero que indica el estado del elemento.

ETIQUETA <ALARMA>

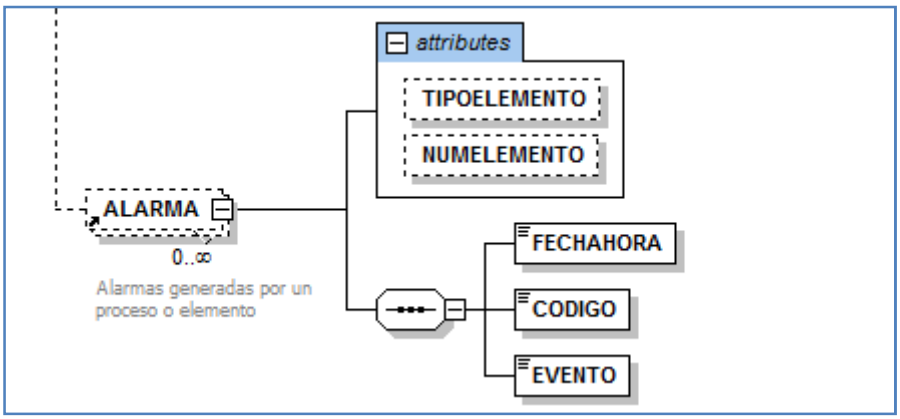

Figura 12: Etiqueta ALARMA

Etiqueta que se puede repetir tantas veces como sea necesario y engloba información relativa a una alarma en un elemento concreto. Para identificar el elemento emplea 2 atributo obligatorios:

- **TIPOELEMENTO:** atributo de valor entero que indica el tipo de elemento. Este campo se corresponde con alguno de los elementos dados de alta en la configuración de la tabla DESC_ELEMENTO descrito en el apartado referente al Modelo de Datos.
- **NUMELEMENTO** atributo de valor entero que indica el número de elemento. Se refiere al identificador del elemento entre varios del mismo tipo (con el mismo TIPOELEMENTO).

Los datos relativos a la alarma se almacenan en las siguientes etiquetas:

- **<FECHAHORA>** Fecha y hora en la ocurrió la alarma.
- **<CODIGO>** Entero que representa el código de alarma. Este campo se corresponde con alguno de los elementos dados de alta en la configuración de la tabla DESC_ALARMA descrito en el apartado referente al Modelo de Datos.
- **<EVENTO>** Entero que representa el evento de la alarma. Este evento puede indicar inicio o fin de alarma. Este campo se corresponde con alguno de los elementos dados de alta en la configuración de la tabla DESC_EVENTO_ALARMA descrito en el apartado referente al Modelo de Datos.

6.3.1.3 EJEMPLO DE MENSAJE DE VÍA

En este apartado se va a mostrar un ejemplo de lo que podría ser un mensaje generado por la vía. De esta forma se podrá analizar de qué manera la información generada por la vía es encapsulada y enviada.

Como ya se ha explicado, este mensaje será traducido por el servidor en tiempo real en un mensaje encapsulado según un esquema diseñado para el manejo de la aplicación web. En el apartado siguiente se mostrará la misma información encapsulada en dicho esquema.

```
<?xml version="1.0" encoding="UTF-8"?>
<MENSAJE IDPROCESO="1" ORIGEN="0x01010101" DESTINO="0xFFFFFFFF"
FECHAHORA="2008-03-10T17:38:38.0+01:00">
<INFO>

      <ESTADOVIA>
            <MODO>1</MODO>              <!– MODO DE APERTURA MANUAL -->
            <FUNCION>2</FUNCION>        <!– VIA ABIERTA DE SALIDA -->
      </ESTADOVIA>

      <TURNO>
            <ESTADO>1</ESTADO>                  <!– VIA EN ESTADO ABIERTO -->
            <FECHAHORA>2008-11-28T13:44:07.0+01:00</FECHAHORA>
                                                <!– FECHA DE APERTURA DEL TURNO -->
            <NUMERO>156</NUMERO>                <!– NUMERO DE TURNO -->
            <IDPEAJISTA>PJ208</IDPEAJISTA>      <!– IDENTIFICADOR DEL PEAJISTA -->
            <NOMBRE>Peajista 01</NOMBRE>        <!– NOMBRE DEL PEAJISTA -->
      </TURNO>

      <COLA NUMVEHICULOS="1"/>   <!– NUMERO DE VEHICULOS EN COLA -->

      <!– ESTADO DEL ELEMENTO ASPA / FLECHA -->
      <ESTADO TIPOELEMENTO="2" NUMELEMENTO="1">
            <FECHAHORA>2008-03-10T11:12:23.0+01:00</FECHAHORA>
            <CODIGOACT>1</CODIGOACT>
      </ESTADO>

      <!– ALARMA EN EL ELEMENTO BARRERA DE ENTRADA -->
      <ALARMA TIPOELEMENTO="01" NUMELEMENTO="01">
             <FECHAHORA>2008-09-19T12:23:20.0+01:00</FECHAHORA>
             <CODIGO>50</CODIGO>
             <EVENTO>1</EVENTO>
      </ALARMA>

      <!– ALARMA EN EL ELEMENTO ANTENA -->
      <ALARMA TIPOELEMENTO="09" NUMELEMENTO="01">
             <FECHAHORA>2008-09-19T12:23:20.0+01:00</FECHAHORA>
             <CODIGO>03</CODIGO>
             <EVENTO>1</EVENTO>
      </ALARMA>

</INFO>
</MENSAJE>
```

6.3.2 MENSAJE_WEB.XSD

La estructura del MensajeWeb.xml está pensada para proporcionar información a la interfaz web en los distintos niveles de detalle que el interfaz de usuario requiere.

Este apartado describe la estructura de los mensajes que se envían a la interfaz web para que esta represente al usuario la información. Estos mensajes se transmiten desde el Servidor en Tiempo Real que mantiene el estado del peaje hacia el servidor web bajo demanda de este.

El Servidor en Tiempo Real es el encargado de parsear los mensajes encapsulados según el esquema descrito en el apartado anterior (**Mensaje.xsd**) y realizar la traducción al esquema **"MensajeWeb.xsd"** que maneja la aplicación web de Supervisión. Este módulo guarda un mapa completo de todo el sistema de peaje encapsulado ya en etiquetas que obedecen a este esquema para reducir al máximo el tiempo de respuesta a la aplicación.

Cuando el interfaz web necesita mostrar información, se la solicita al Servidor en Tiempo Real mediante una petición HTTP a un puerto configurado a tal efecto y el Servidor en Tiempo Real devolverá un fichero XML encapsulado según el esquema MensajeWeb.xsd con la información solicitada.

6.3.2.1. VISIÓN GLOBAL

La intercomunicación que aquí se describe solo tiene lugar entre dos elementos, que son, el Servidor en Tiempo Real que mantiene la información, en tiempo real, del estado del peaje y la aplicación web que se encarga de presentarla al usuario.

El Servidor en Tiempo Real además de mantener el estado del peaje, implementa un servidor web a través del cual enviará la información.

Existen varios tipos de peticiones implementas por el Servidor en Tiempo Real que responden a diferentes grados de detalle. Una explicación detallada de las peticiones implementadas por el Servidor en Tiempo Real se explicará en el apartado de Desarrollo de la Aplicación. De momento nos vale con saber que fundamentalmente existen 2 tipos de petición:

- Una petición http que lleva un parámetro **"id"** que indica el lugar del cual se quiere extraer la información.

- Una petición extendida que pide que se le devuelvan todas las alarmas que hay en el sistema en un momento dado. Esta petición lleva el literal **"alarmas"**.

El primer tipo de petición implica tres niveles de detalle según cuál sea el ID_LUGAR solicitado:

- Cuando se soliciten datos de una vía se devolverán todos los datos que el Servidor en Tiempo Real tenga sobre esa vía en concreto.
- Cuando la solicitud de datos sea acerca de una estación el módulo devolverá toda la información que se tenga sobre la estación propiamente dicha, así como información básica sobre las vías que dependan de esa estación. Esta información básica está compuesta de estado, función y modo de la vía, numero de vehículos en la vía y si esta tiene activa alguna alarma.
- Cuando la solicitud de datos sea acerca del centro de control el módulo devolverá toda la información que posea sobre el centro de control propiamente dicho, así como información básica sobre las estaciones y la información básica sobre todas la vías. La información de las estaciones será el estado de la estación y si tiene alguna alarma o no. En este caso una estación tendrá alarma si ella misma tiene alguna alarma o si alguna de sus vías tiene alguna alarma.

6.3.2.2. DESCRIPCIÓN DE LOS TAGS

La Figura 13 representa un diagrama que muestra los TAGS que componen este mensaje. A continuación se pasa a describirlos uno a uno.

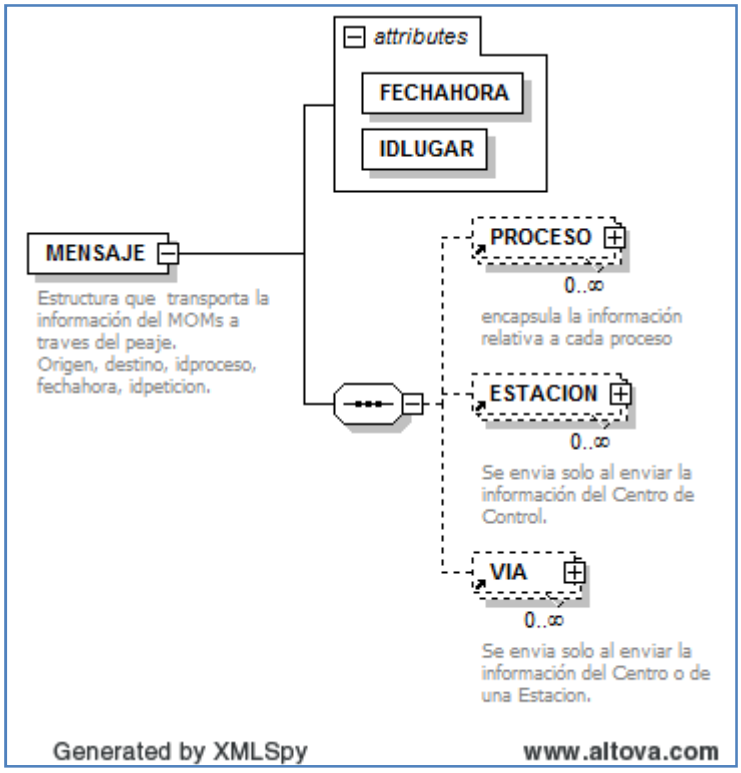

Figura 13: Mensaje de información web

ETIQUETA <MENSAJE>

Esta es la etiqueta principal y engloba todo el mensaje. Sólo puede haber una por mensaje y ha de llevar los siguientes atributos:

- **FECHAHORA**: Fecha y hora en la que se genera el mensaje.
- **IDLUGAR**: Entero que indica el idlugar del que se ha solicitado información. Coincidirá con el idlugar de la petición.

La etiqueta **MENSAJE** envía la información desglosada por procesos. Esta incluye una lista de etiquetas **PROCESO** cada una de las cuales encapsula la información relativa a su proceso propiamente dicho. Cuando se solicita información al nivel de vía, el Servidor en Tiempo Real envía la información referente a esa vía encapsulada en esta etiqueta identificada por el **ID_PROCESO** correspondiente al controlador de vía según lo configurado en la tabla DESC_PROCESO.

En caso de que se solicite información al nivel de detalle de estación, se enviarán tantas etiquetas VIA como vías de esa estación tenga información el Servidor en Tiempo Real. El nivel de detalle que se da de las vías encapsulado en esta etiqueta es menor que el que proporciona de la vía el tag PROCESO.

Cuando el nivel de detalle solicitado es de Centro de Control, se enviarán tantas etiquetas ESTACION como estaciones de ese Centro tenga información el Servidor en Tiempo Real, así como tantas etiquetas VIA como vías de ese Centro tenga información.

A continuación se pasa a concretar cuál es la información que encapsula cada una de estas etiquetas.

ETIQUETA <PROCESO>

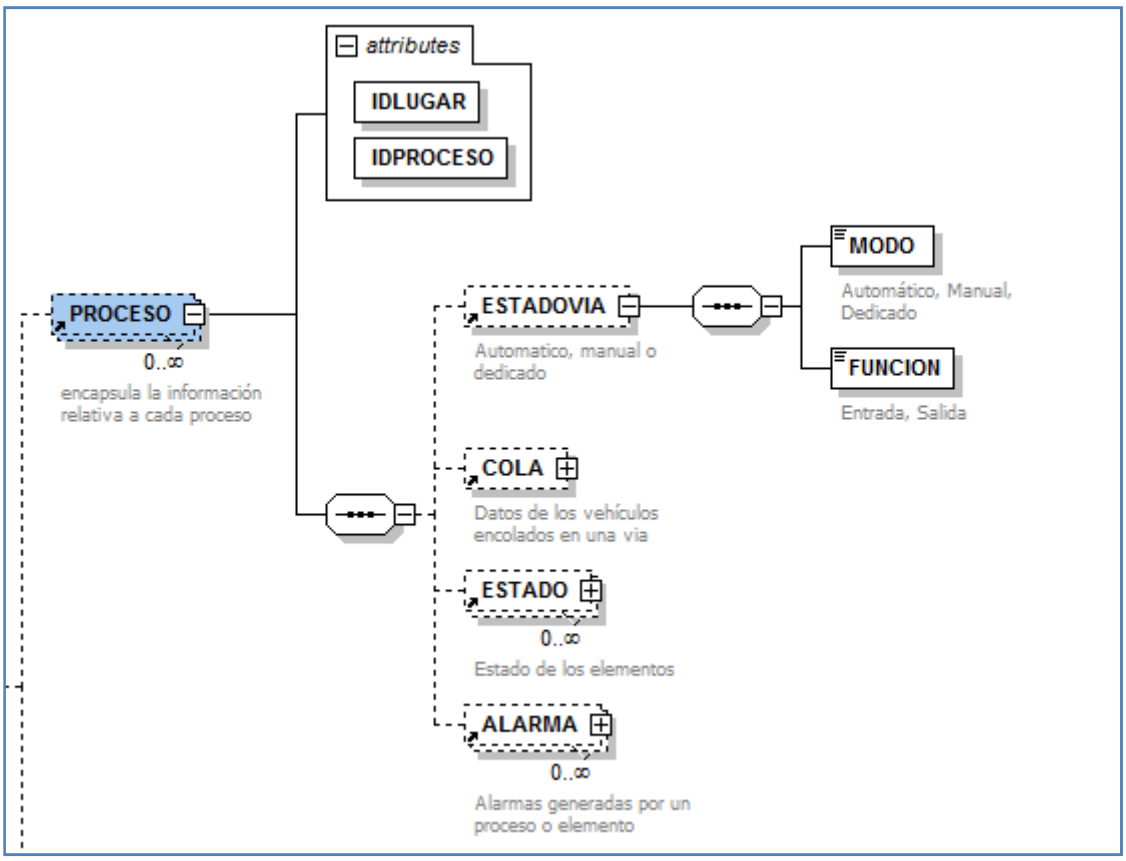

Figura 14: Etiqueta PROCESO

Contiene la información relativa a un proceso que se ejecuta en un idLugar. Para identificar esto emplea los siguientes atributos:

- **IDLUGAR**: Entero que indica el idLugar del que se envía la información.
- **IDPROCESO**: Entero con el identificador de proceso del que se envía la información.

Esta etiqueta contendrá una combinación, dependiendo de la petición que se haya realizado, de las etiquetas que se muestran a continuación. Pueden aparecer o no, pero siempre lo harán en el orden en el que se presentan aquí.

ETIQUETA <ESTADOVIA>

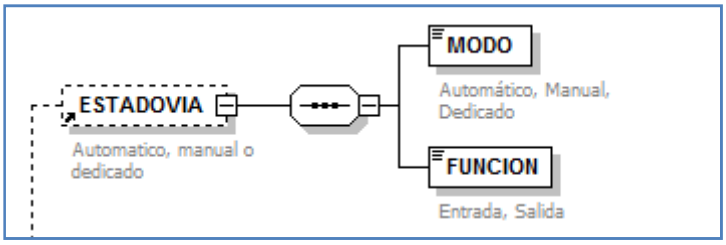

Figura 15: Etiqueta ESTADOVIA

Datos de modo y función en el que opera una vía. Esta solo se enviará cuando se soliciten datos de una vía en concreto.

- **<MODO>** Entero que describe el modo en que funciona la vía. Este código se corresponde con alguno de los identificadores configurados en la tabla DESC_MODO_VIA descrita en el apartado referente al Modelo de Datos.

Según este mismo apartado, los posibles modos en que opera la vía son:

1. Manual
2. Automático
3. Telepeaje

- **<FUNCION>** Entero que indica si la vía está abierta como vía de entrada o como vía de salida. Este código se corresponde con alguno de los identificadores configurados en la tabla DESC_FUNCION_VIA descrita en el apartado referente al Modelo de Datos.

1. Entrada: funciona como tiquetera.
2. Salida: salida del peaje, cobra.

ETIQUETA <COLA>

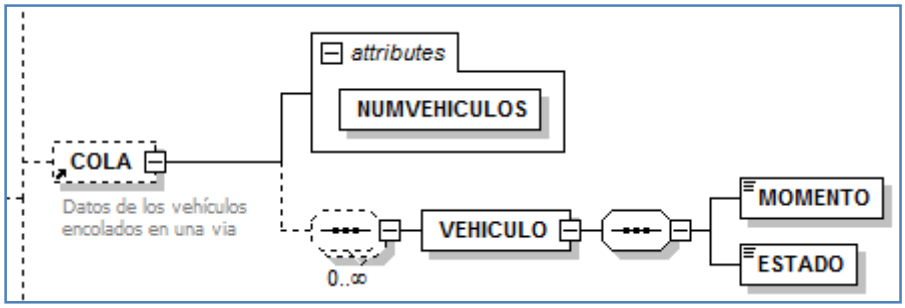

Figura 16: Etiqueta COLA

Datos sobre los vehículos existentes en una vía. Esta etiqueta contendrá una secuencia de una o más etiquetas **VEHICULO**.

- **<VEHICULO>** Contiene datos de un vehículo en una cola. Muestra el estado de cada vehículo y su posición en el sistema de clasificación de la vía.
- **<MOMENTO>** Entero que indica la posición del vehículo en el sistema de clasificación de la vía
- **<ESTADO>** Entero que indica el estado de ese vehículo.

ETIQUETA <ESTADO>

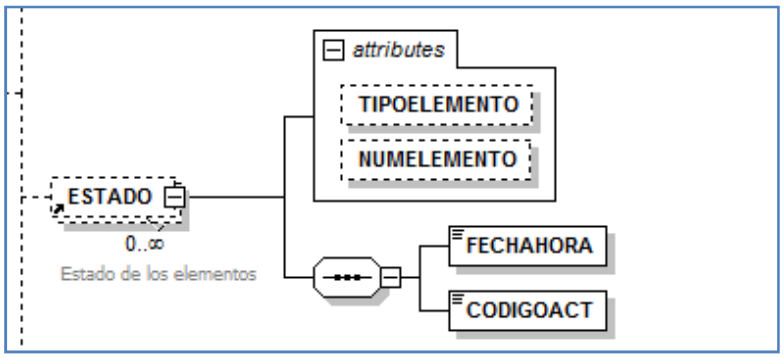

Figura 17: Etiqueta ESTADO

Datos del estado de los elementos. Etiqueta que se puede repetir tantas veces como sea necesario, que agrupa la información del estado de un elemento concreto que identifica con dos atributos obligatorios:

- **TIPOELEMENTO:** atributo de valor entero que indica el tipo de elemento.
- **NUMELEMENTO:** atributo de valor entero que indica el número de elemento.

La información sobre el estado del elemento se representa con las etiquetas:

- **<FECHAHORA>:** Fecha y hora asociadas al estado del elemento.
- **<CODIGOACT>:** Entero que indica el estado del elemento.

ETIQUETA <ALARMA>

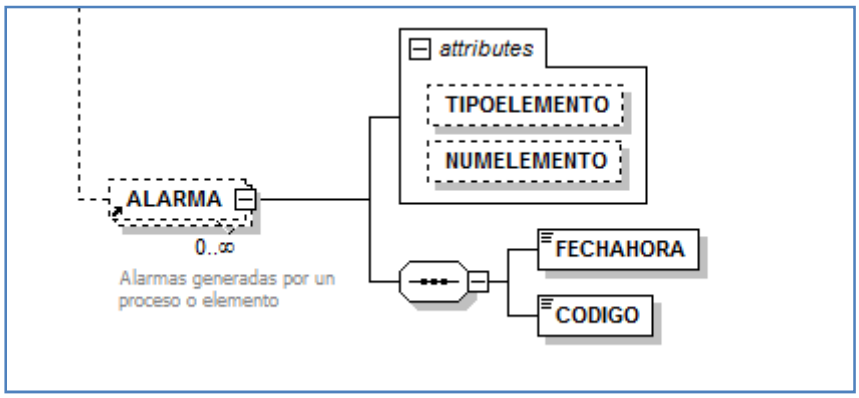

Figura 18: Etiqueta ALARMA

Datos de las alarmas. Se presentará tantas veces como alarmas existan en el nivel solicitado (vía, estación o centro) y llevará la información de la alarma correspondiente. Para identificar el elemento emplea 2 atributos obligatorios:

- **TIPOELEMENTO:** atributo de valor entero que indica el tipo de elemento.
- **NUMELEMENTO:** atributo de valor entero que indica el número de elemento.

Los datos relativos a la alarma se almacenan en las siguientes etiquetas:

- **<FECHAHORA>:** Fecha y hora en la ocurrió la alarma.
- **<CODIGO>**: Entero que representa el código de alarma.

ETIQUETA <VIA>

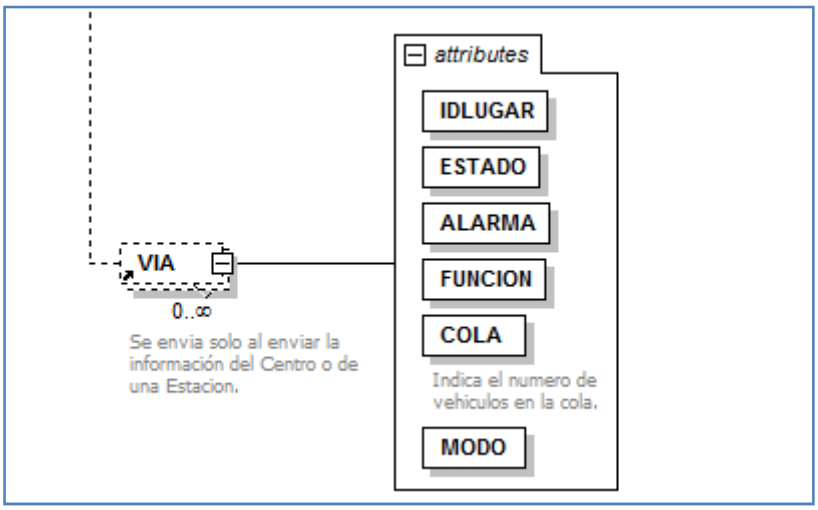

Figura 19: Etiqueta VIA

Resumen de datos de una vía. Esta etiqueta se presentará tantas veces como vías se vayan a representar. Básicamente la información que lleva es el estado y modo de funcionamiento de la vía, el número de vehículos existentes en la vía y si esta vía tiene alguna alarma o no. Esta etiqueta de generará tan solo al solicitar datos de una estación o del centro de control.

ETIQUETA <ESTACION>

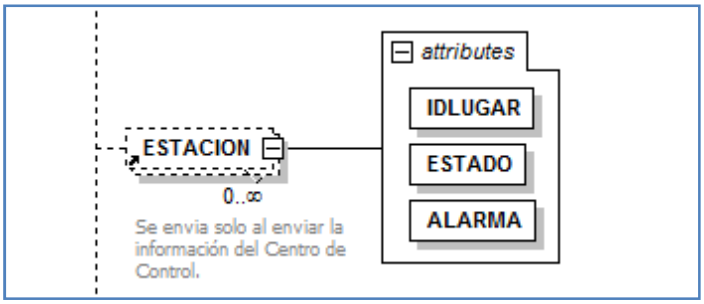

Figura 20: Etiqueta ESTACION

Resumen de los datos de una estación. Se presentará tantas veces como estaciones haya y contendrá el estado de la estación y si tiene alguna alarma o no. Esta etiqueta se enviará sólo al solicitar los datos del centro de control.

6.3.2.1 EJEMPLOS DE MENSAJES CON ESQUEMA WEB

En este apartado se van a mostrar los mensajes manejados por la web en cada uno de los tres niveles. De esta forma se puede apreciar mejor la filosofía que se ha seguido a la hora de diseñar el esquema y porqué este es fácilmente manejable por la web.

NIVEL DE DETALLE DE VÍA

```
<?xml version="1.0" encoding="UTF-8"?>
<MENSAJE IDLUGAR="16843009" FECHAHORA="2009-12-25T10:45:07.744+01:00">

<PROCESO IDPROCESO="1" IDLUGAR="16843009">

      <ESTADOVIA>
            <MODO>1</MODO>              <!— MODO DE APERTURA MANUAL -->
            <FUNCION>2</FUNCION>        <!— VIA ABIERTA DE SALIDA -->
      </ESTADOVIA>

      <TURNO>
            <ESTADO>1</ESTADO>                  <!— VIA EN ESTADO ABIERTO -->
            <FECHAHORA>2008-11-28T13:44:07.0+01:00</FECHAHORA>
                                               <!— FECHA DE APERTURA DEL TURNO -->
            <NUMERO>156</NUMERO>                <!— NUMERO DE TURNO -->
            <IDPEAJISTA>PJ208</IDPEAJISTA>      <!— IDENTIFICADOR DEL PEAJISTA -->
            <NOMBRE>Peajista 01</NOMBRE>        <!— NOMBRE DEL PEAJISTA -->
      </TURNO>

      <COLA NUMVEHICULOS="1"/>   <!— NUMERO DE VEHICULOS EN COLA -->

      <!— ESTADO DEL ELEMENTO ASPA / FLECHA -->
      <ESTADO TIPOELEMENTO="2" NUMELEMENTO="1">
            <FECHAHORA>2008-03-10T11:12:23.0+01:00</FECHAHORA>
            <CODIGOACT>1</CODIGOACT>
      </ESTADO>

      <!— ALARMA EN EL ELEMENTO BARRERA DE ENTRADA -->
      <ALARMA TIPOELEMENTO="01" NUMELEMENTO="01">
            <FECHAHORA>2008-09-19T12:23:20.0+01:00</FECHAHORA>
            <CODIGO>50</CODIGO>
            <EVENTO>1</EVENTO>
      </ALARMA>

      <!— ALARMA EN EL ELEMENTO ANTENA -->
      <ALARMA TIPOELEMENTO="09" NUMELEMENTO="01">
            <FECHAHORA>2008-09-19T12:23:20.0+01:00</FECHAHORA>
            <CODIGO>03</CODIGO>
            <EVENTO>1</EVENTO>
      </ALARMA>

</PROCESO>
</MENSAJE>
```

NIVEL DE DETALLE DE ESTACIÓN

```
<?xml version="1.0" encoding="UTF-8"?>
<MENSAJE IDLUGAR="16843008" FECHAHORA="2009-12-25T10:53:18.450+01:00">
<ESTACION ALARMA="1" ESTADO="0" IDLUGAR="16843008"/>
<ESTACION ALARMA="0" ESTADO="0" IDLUGAR="16843264"/>
<ESTACION ALARMA="1" ESTADO="0" IDLUGAR="16843520"/>
<VIA SMODO="-1" MODO="1" COLA="1" FUNCION="2" ALARMA="1" ESTADO="1" IDLUGAR="16843010"/>
<VIA SMODO="-1" MODO="2" COLA="3" FUNCION="2" ALARMA="0" ESTADO="3" IDLUGAR="16843011"/>
<VIA SMODO="-1" MODO="3" COLA="5" FUNCION="2" ALARMA="0" ESTADO="1" IDLUGAR="16843009"/>
<VIA SMODO="-1" MODO="3" COLA="1" FUNCION="2" ALARMA="1" ESTADO="1" IDLUGAR="16843012"/>
<VIA SMODO="-1" MODO="1" COLA="1" FUNCION="2" ALARMA="0" ESTADO="2" IDLUGAR="16843013"/>
</MENSAJE>

      <!— MODO: 1: MANUAL / 2: AUTOMÁTICO / 3: DINÁMICO -->
      <!— FUNCION: 1: ENTRADA / 2: SALIDA -->
```

NIVEL DE DETALLE DE CENTRO

```
<?xml version="1.0" encoding="UTF-8"?>
<MENSAJE IDLUGAR="16843008" FECHAHORA="2009-12-25T10:53:18.450+01:00">
<ESTACION ALARMA="1" ESTADO="0" IDLUGAR="16843008"/>
<ESTACION ALARMA="0" ESTADO="0" IDLUGAR="16843264"/>
<ESTACION ALARMA="1" ESTADO="0" IDLUGAR="16843520"/>
<VIA SMODO="-1" MODO="1" COLA="1" FUNCION="2" ALARMA="1" ESTADO="1" IDLUGAR="16843010"/>
<VIA SMODO="-1" MODO="2" COLA="3" FUNCION="2" ALARMA="0" ESTADO="3" IDLUGAR="16843011"/>
<VIA SMODO="-1" MODO="3" COLA="5" FUNCION="2" ALARMA="0" ESTADO="1" IDLUGAR="16843009"/>
<VIA SMODO="-1" MODO="3" COLA="1" FUNCION="2" ALARMA="1" ESTADO="1" IDLUGAR="16843012"/>
<VIA SMODO="-1" MODO="1" COLA="1" FUNCION="2" ALARMA="0" ESTADO="2" IDLUGAR="16843013"/>
</MENSAJE>

      <!— MODO: 1: MANUAL / 2: AUTOMÁTICO / 3: DINÁMICO -->
      <!— FUNCION: 1: ENTRADA / 2: SALIDA -->
```

7 MODELO DE DATOS

El sistema de peaje tiene motor de base de datos en todos los niveles de su arquitectura. Como ya se ha explicado anteriormente, el sistema de supervisión y monitorización web tiene su propio motor de Base de Datos que contiene tanto información de históricos de alarmas del sistema susceptibles de ser consultados por medio de la aplicación, como información de configuración de la aplicación, como otro tipo de información necesaria para el correcto funcionamiento del sistema.

El presente apartado tiene como objetivo la explicación del diseño de la base de datos que subyace a la aplicación web de Supervisión. Se explicarán el significado y diseño de todas las tablas así como el modelo de interrelación de datos en los módulos más importantes de la aplicación en lenguaje UML.

7.1.VISTA GENERAL DE LA BASE DE DATOS

Las tablas que constituyen la base de datos pueden agruparse lógicamente según la función que desempeñen. En este apartado se describe completamente el modelo lógico de organización de los datos separado en tres apartados según la función que desempeñan las tablas. Existen según esta clasificación 3 tipos de tablas: tablas de históricos o sucesos, tablas de descripción o lectura y tablas de configuración del sistema.

7.1.1 TABLAS DE HISTÓRICOS

Registran individualmente los sucesos que van teniendo lugar en las vías y en cada uno de sus elementos. Hay una tabla distinta para cada tipo de suceso. Hay una serie de campos con información general común a cualquier tipo de suceso que existe en todas las tablas. Los sucesos se van guardando en registros sucesivos.

En las tablas de sucesos queda guardada tanto la historia de los sucesos acontecidos en las vías como cualquier tipo de fallo o alarma que se produzca en algún punto del sistema. Los registros pertenecientes a

estas tablas deben tener un campo que identifique el momento en que se produjo, así como otra serie de campos necesarios para la perfecta e inequívoca definición del suceso.

Estas tablas van creciendo a gran velocidad, por lo que deben ser objeto de una política de mantenimiento regular.

7.1.1.1 LISTADO DE TABLAS

- H_ALARMA

7.1.1.2 DESCRIPCIÓN DE LAS TABLAS

A continuación se explica el diseño de las tablas de históricos especificando el nombre y significado de sus campos, así como cuales de estos campos componen una **Primary Key** (señalado como PK) o **Foreign Key** (señalado como FK) de otra tabla.

TABLA H_ALARMA

Esta tabla almacena información sobre todas las alarmas y fallos que se producen en el sistema de peaje. Cada vez que ocurre una alarma en el peaje y esta sube al Centro de Control, un proceso se encargará de insertar un registro en esta tabla indicando cuando se produjo dicha alarma, donde se produjo, cual es el dispositivo que generó la alarma y qué tipo de alarma presenta. Además quedará registrado el momento en que se generó dicha alarma así como el instante en que esta dejó de detectarse.

Esta tabla es extremadamente importante puesto que a partir de los datos insertados en esta tabla se podrán realizar análisis posteriores, así como generar informes y estudios que permitan realizar un mantenimiento predictivo y preventivo de los errores del sistema.

Nombre del campo	Tipo de Dato	Descripción	Referencia a otras tablas
FECHA_HORA *	DATETIME NOT NULL	Fecha del suceso	
ZONA_HORARIA	INT	Zona horaria	
ID_LUGAR *	INT NOT NULL	Lugar donde se genera la alarma	LUGARES
ID_PROCESO *	INT NOT NULL	Proceso controlador del dispositivo	DESC_PROCESO
ID_ELEMENTO *	INT NOT NULL	Dispositivo que genera la Alarma	DESC_ELEMENTO
N_ELEMENTO *	INT NOT NULL	# de dispositivo	
ID_ALARMA *	INT NOT NULL	Tipo de Alarma	DESC_ALARMA
ID_EVENTO_ALARMA *	INT NOT NULL	Inicio / Fin de Alarma	DESC_EVENTO_ALARMA
ID_PRIORIDAD_ALARMA	INT	Prioridad de la Alarma	DESC_PRIORIDAD_ALARMA
INFO	VARCHAR (3940)	Información adicional de Alarma	
PROCESADO	TINYINT	Flag de procesado	

Como puede observarse, a partir de un registro de estas tablas queda completa y perfectamente definida una alarma producida en el sistema:

- Se indica cuando se produjo el evento mediante el campo FECHA_HORA.
- El campo ID_LUGAR describe en qué lugar del peaje se produjo. Este campo está relacionado con la Primary Key de la tabla LUGARES que describo más adelante.
- El proceso que falló o que controla el dispositivo que falló está definido mediante el elemento ID_PROCESO relacionado con la tabla DESC_PROCESO que describimos más adelante.
- El dispositivo que generó la alarma viene definido por el campo ID_ELEMENTO directamente relacionado con la tabla DESC_ELEMENTO.
- El número de elemento en cuestión es el campo N_ELEMENTO (pueden haber 2 barreras, 2 semáforos, varias antenas…)

- El campo ID_EVENTO_ALARMA relacionado con la tabla DESC_EVENTO_ALARMA indica si el evento es de inicio o de fin de alarma. De esta manera tenemos un histórico completo de todos los eventos. Sabemos la duración de las alarmas.

Es importante advertir que esta configuración deja abiertas todas las posibilidades y combinaciones entre tipos de alarma (ID_ALARMA) y dispositivos (ID_ELEMENTO). Es función de los procesos que generan estas alarmas introducir los códigos correctos para que la información mostrada al usuario en la interfaz web sea coherente, pero en ningún caso la generación de alarmas incoherentes desencadenará un error en la aplicación web. Esta se limitará, como ya he apuntado anteriormente, a mostrar al usuario la información que se le proporciona apoyándose en la configuración que se haya llevado a cabo en el resto de tablas relacionadas y que se describen a continuación.

7.1.2 TABLAS DE DESCRIPCIÓN

En general este tipo de tablas asocian a determinados códigos su descripción. Todas estas tablas tienen dos campos: un campo numérico (el código) y un campo de texto (la descripción). En el resto de tablas asociadas únicamente se almacenará el código, mientras que el interfaz web utilizará estas tablas de descripción para mostrar información descriptiva y fácil de entender por el usuario.

Cuando se quieren incluir nuevos elementos o procesos susceptibles de ser monitorizados por el sistema, no hay más que insertar nuevos registros en estas tablas, de manera que el sistema puede crecer de manera completamente transparente a la aplicación web que se limita a leer datos y mostrárselos al usuario según la configuración almacenada en las tablas correspondientes de la base de datos, pero en ningún caso interpreta esos datos.

Ni la información ni el tamaño de estas tablas cambiará en periodos largos de tiempo en la mayoría de los casos.

7.1.2.1 LISTADO DE TABLAS

- DESC_ALARMA
- DESC_ELEMENTO
- DESC_PROCESO
- DESC_EVENTO_ALARMA
- DESC_PRIORIDAD_ALARMA
- DESC_MODO_VIA
- DESC_FUNCION_VIA

7.1.2.2 DESCRIPCIÓN DE LAS TABLAS

A continuación se indican las tablas de descripción que se han empleado para la aplicación web.

TABLA DESC_ALARMA

Esta tabla almacena los pares código-descripción que describe los distintos tipo de alarmas.

Nombre del Campo	Tipo de Dato	Descripción
ID_ALARMA *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA DESC_ELEMENTO

Esta tabla almacena los pares código-descripción que describe los distintos tipo elementos de la vía.

Nombre del Campo	Tipo de Dato	Descripción
ID_ELEMENTO *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA_DESC_PROCESO

Esta tabla almacena los pares código-descripción que describe los distintos procesos del sistema.

Nombre del Campo	Tipo de Dato	Descripción
ID_PROCESO *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA_DESC_EVENTO_ALARMA

Esta tabla almacena los pares código-descripción correspondientes al evento de inicio y al fin de alarma.

Nombre del Campo	Tipo de Dato	Descripción
ID_EVENTO_ALARMA *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA_DESC_PRIORIDAD_ALARMA

Esta tabla almacena los pares código-descripción que describe la prioridad de las alarmas. La prioridad de las alarmas las configura el usuario desde el módulo de "Configuración de alarmas". Esta configuración se almacena en la tabla C_ALARMA que se describe más adelante. La prioridad de la alarma define el tratamiento posterior que debe desencadenar esta alarma.

Nombre del Campo	Tipo de Dato	Descripción
ID_PRIORIDAD_ALARMA *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA DESC_MODO_VIA

Esta tabla almacena los pares código-descripción correspondientes al modo de operación de la vía.

Nombre del Campo	Tipo de Dato	Descripción
ID_MODO_VIA *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA DESC_FUNCION_VIA

Esta tabla almacena los pares código-descripción correspondientes a vía de entrada o de salida.

Nombre del Campo	Tipo de Dato	Descripción
ID_FUNCION_VIA *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

7.1.3 TABLAS DE CONFIGURACIÓN

Estas tablas definen la configuración de determinados parámetros del sistema y que definen su funcionamiento. Estas tablas tienen por lo general un tamaño constante y se van actualizando ciertos flags en función de la configuración escogida por el usuario por medio de las distintas funcionalidades que ofrece la aplicación web.

7.1.3.1 LISTADO DE TABLAS

- LUGARES
- USUARIOS
- ROLES
- ROLES_USUARIOS

7.1.3.2 DESCRIPCIÓN DE LAS TABLAS

TABLA LUGARES.

La tabla LUGARES contiene todas las ubicaciones del peaje. Cualquier tabla que contenga un registro que haga referencia a alguna ubicación del peaje, se relaciona con esta tabla.

Esta tabla contiene una “Primary Key” única llamada **ID_LUGAR** que identifica de forma unívoca a cada una de las ubicaciones del peaje. ID_LUGAR corresponde en realidad con el equivalente en decimal de una construcción hexadecimal que identifica el centro, la estación y la vía a la que pertenece la ubicación. El ID_LUGAR en hexadecimal sería algo así: 0xCCSSEEVV, donde:

- CC: dos dígitos que identifican el Centro de Control
- SS: siempre se pone a 01. Se reserva para un posible uso futuro
- EE: dos dígitos para el identificador de estación
- VV: dos dígitos para el identificador de vía

A su vez, cada uno de estos identificadores se corresponde con un campo de la tabla. La tabla está diseñada de la siguiente manera:

Nombre del campo	Tipo de Dato	Descripción
ID_LUGAR *	INT NOT NULL	Identificador único
ZONA	TINYINT NOT NULL	Id de Centro
SUBZONA	TINYINT NOT NULL	Para uso futuro
ESTACION	TINYINT NOT NULL	Id. De Estación
VIA	TINYINT NOT NULL	Id de vía
DESCRIPCION	NVARCHAR (30) NULL	Descripción de la ubicación
HOST_NAME	NVARCHAR (30) NULL	DNS Host Name o IP Address

En los mensajes XML de información y alarmas que envían las vías, tanto los campos ORIGEN como DESTINO están definidos por un ID_LUGAR.

TABLA USUARIOS

La tabla USUARIOS alberga todos los usuarios que tiene acceso a la aplicación.

Nombre del Campo	Tipo de Dato	Descripción
ID_USUARIO *	INT NOT NULL	ID único de usuario
LOGIN	VARCHAR (50) NOT NULL	Login de usuario
PASSWORD	VARCHAR (50) NOT NULL	Psswd encriptada MD5
ENABLED	BIT NOT NULL	Bit habilitación
NOMBRE_USUARIO	VARCHAR (250) NULL	Nombre completo del usuario

TABLA ROLES

La tabla ROLES alberga todos los roles permitidos en la aplicación.

Nombre del Campo	Tipo de Dato	Descripción
ID_ROL *	INT NOT NULL	Código
DESCRIPCION	NVARCHAR (50)	Descripción

TABLA ROLES_USUARIOS

Define las asociaciones de usuarios con roles del sistema.

Nombre del Campo	Tipo de Dato	Descripción	Referencia a otras tablas
ID_USUARIO *	INT NOT NULL	Código	USUARIOS
ID_ROL *	INT NOT NULL	Descripción	ROLES

Cada usuario podrá tener uno a varios roles.

7.2.MODELO DE DATOS DE ALARMAS

La figura 21 describe en lenguaje UML la relación entre las diferentes tablas involucradas en el modelo de Datos de Alarmas.

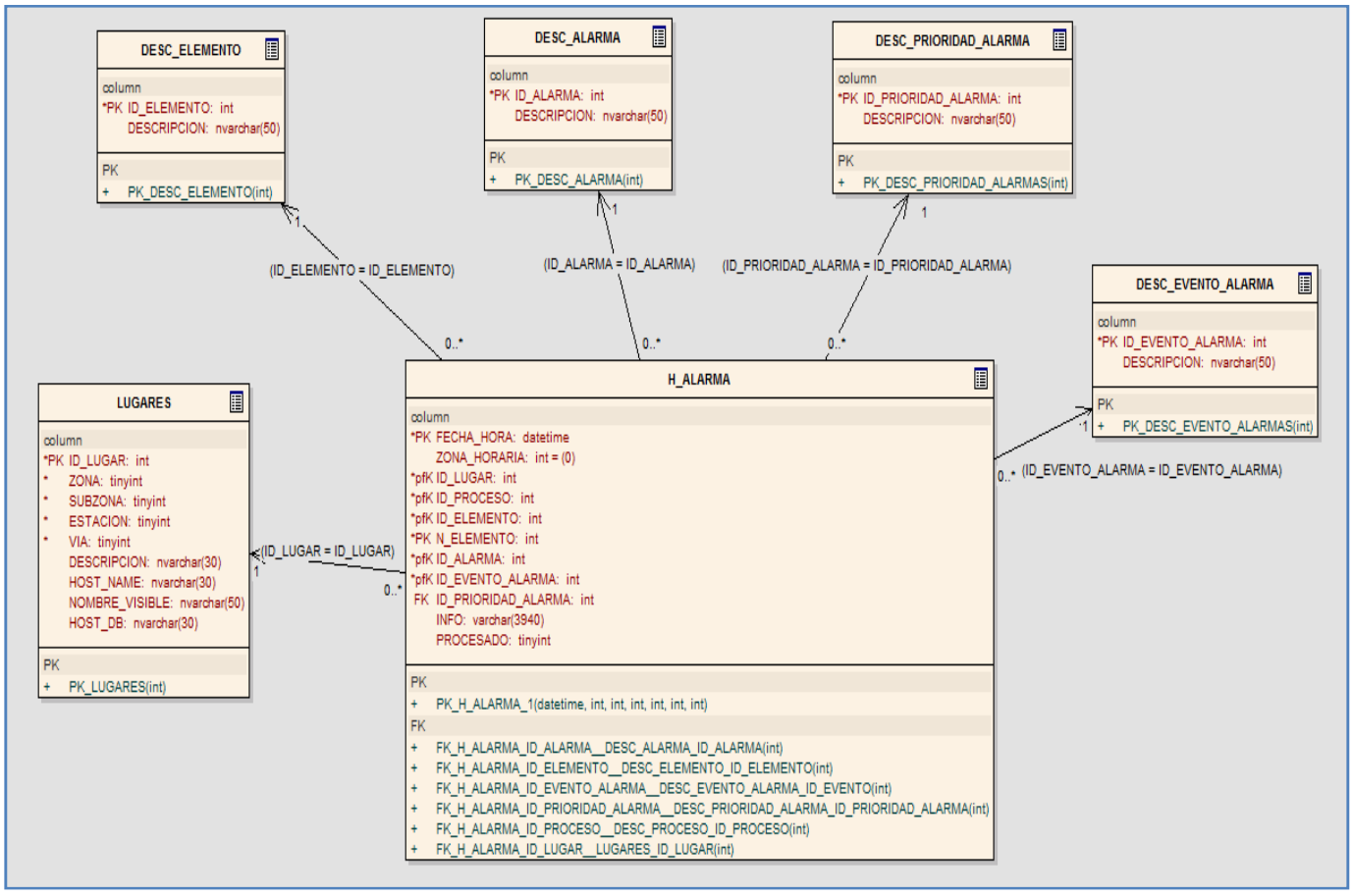

Figura 3: Modelo de datos de alarmas

7.3.MODELO DE DATOS DE USUARIOS

La figura 22 describe en lenguaje UML la relación entre las diferentes tablas involucradas en el Modelo de Datos de Usuarios.

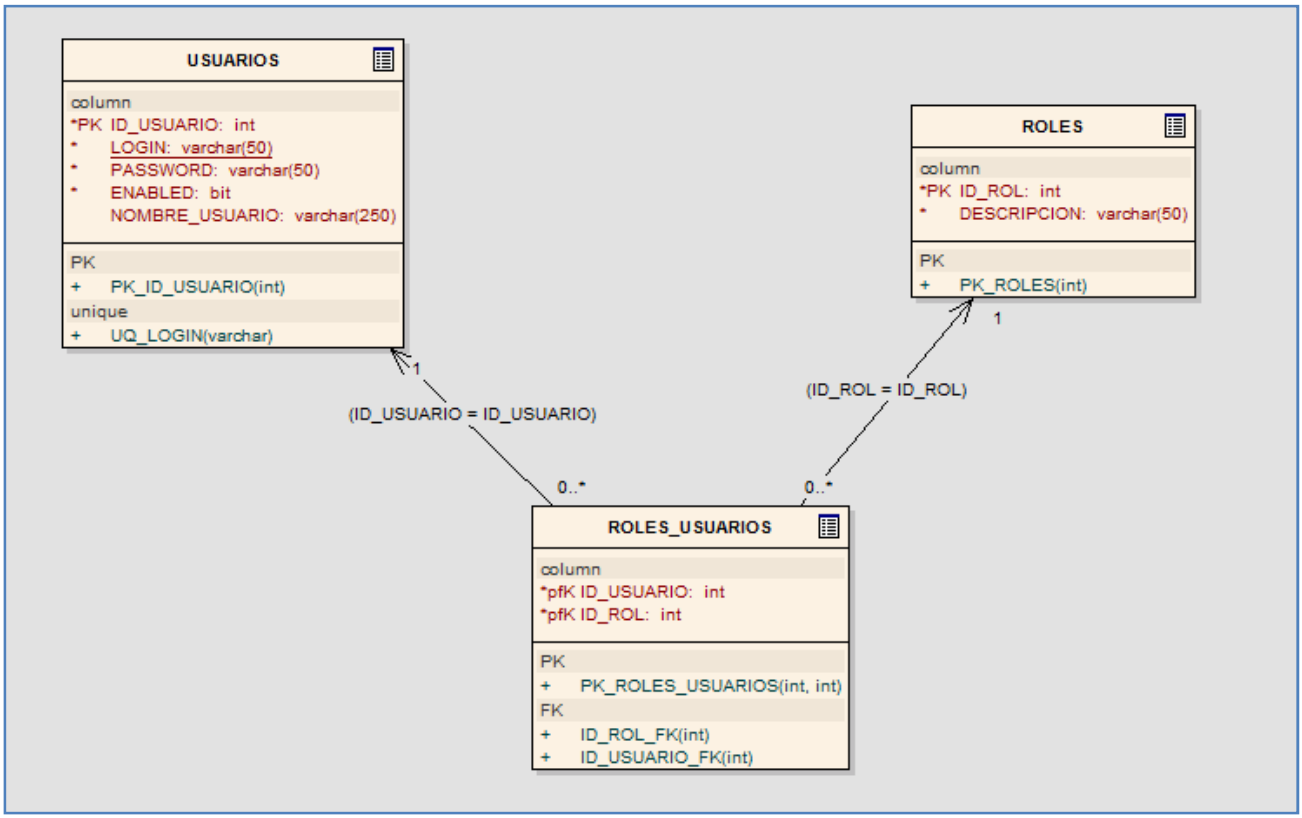

Figura 4: Modelo de datos de usuarios

8 DESCRIPCIÓN FUNCIONAL

La aplicación WEB se divide en dos módulos básicos:

- **Monitorización y visualización**. Este módulo incluye dos submódulos principalmente
  - Monitorización de Centro y de Estación en tiempo real
  - Visualización de Alarmas en Tiempo Real y de históricos
- **Administración y configuración**. Incluye dos submódulos principalmente:
  - Gestión de usuarios: asignación de roles a usuarios del sistema
  - Configuración de la prioridad de las alarmas del sistema

8.1.AUTENTICACIÓN Y CONTROL DE USUARIOS

El acceso a la aplicación requiere autenticación por parte de usuarios habilitados. Además, en función de los roles asociados, al usuario validado en la aplicación se le permitirá el acceso a ciertas partes de la aplicación y se le restringirá el acceso a otras.

En concreto los roles contemplados en la aplicación son los siguientes:

- **Administrador**: permite el acceso a todas las partes de la aplicación.
- **Controlador de Estación**: permite monitorizar las estaciones del peaje.
- **Controlador de Centro**: permite monitorizar el Centro de Control.
- **Visualización de Alarmas:** permite visualizar alarmas en tiempo real e históricos.
- **Configuración:** permite acceder el módulo de administración y configuración.

El Módulo de Gestión de usuarios explicado posteriormente, permite al usuario “Administrador” o con permisos de configuración asignar roles a usuarios.

8.2.MÓDULO DE MONITORIZACIÓN Y VISUALIZACIÓN

Este módulo incluye principalmente las funciones de visualización y monitorización del sistema de peaje.

La funcionalidad de monitorización permite visualizar gráficamente información y estado en tiempo real del peaje tanto a nivel de Centro de Control como a nivel de Estación.

El Módulo de Visualización de Alarmas, incluye dos funcionalidades. La primera monitoriza en tiempo real las alarmas del sistema, mientras que la segunda permita consultar históricos de alarmas almacenados en Base de Datos mediante filtros que el interfaz pone a disposición del usuario para tal efecto.

8.2.1 MONITORIZACIÓN DE CENTRO DE CONTROL

A través de este módulo el usuario podrá visualizar en modo gráfico el estado de todo el peaje representado sobre el mapa completo de la autopista extraído gracias al API proporcionado por Google Maps.

Concretamente la autopista discurre a lo largo de poco menos de 200 millas desde Chicago en el estado de Illinois atravesando el estado de Indiana hasta aproximadamente su frontera con el estado de Ohio. Está dividida en dos fases que pueden visualizarse de manera independiente en el mapa mediante unas cajas de selección habilitadas a tal efecto.

En el mapa están marcadas todas las estaciones del peaje. Los marker de las estaciones se distinguirán por un color rojo cuando la estación en cuestión tenga algún tipo de alarma, mientras que tendrán un color verde cuando no tenga ninguna alarma.

Además cada “marker” tendrá asociada una viñeta con doble pestaña. En la primera se podrá visualizar información en tiempo real acerca del modo en que están abiertas las vías de la estación en cuestión, mientras que la segunda ofrecerá información en tiempo real acerca de las alarmas generadas en esta estación en caso de que las haya.

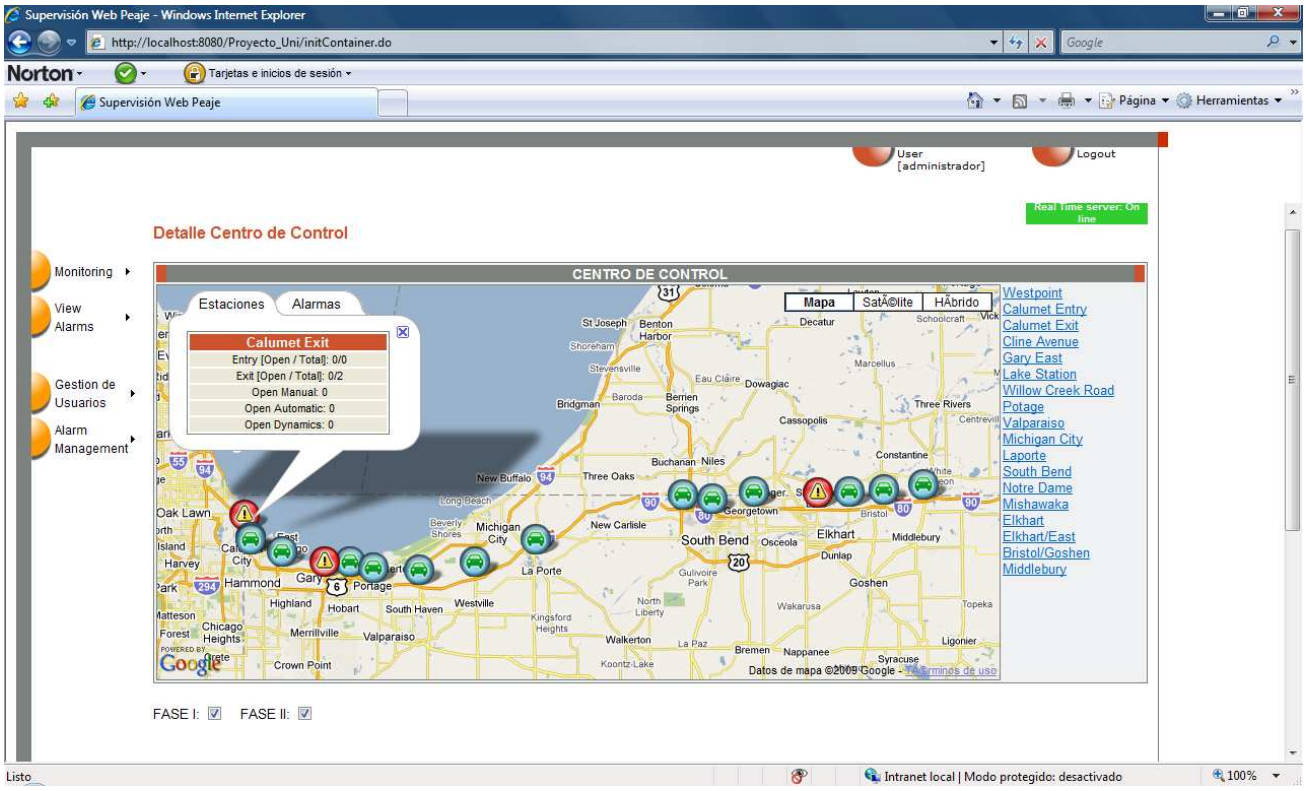

Figura 5: Monitorización de Centro de Control

Por otra parte, se mostrará a elección del usuario una barra lateral a la izquierda del mapa con los enlaces de cada una de las estaciones que permitirá acceder a la funcionalidad de monitorización al nivel de estación explicado en el siguiente apartado. La selección de la Fase en las cajas de texto condicionará la aparición o desaparición de las correspondiente estaciones del peaje.

También se podrá hacer uso de la funcionalidad proporcionada por Google Maps para visualizar el mapa en modo Satélite o Híbrido o para aplicar zoom a la imagen.

Para acceder a la funcionalidad de Monitorización de Centro de Control, será necesario que el usuario tengo el rol de Controlador de Centro o de Administrador.

8.2.2 MONITORIZACIÓN DE ESTACIÓN

Cuando bajemos al nivel de estación se mostrará un layout representando la playa de vías de dicha estación. En la playa de peaje, se mostrará gráficamente la siguiente información para cada una de las vías:

- Tipo de vía: entrada o salida (representado con flecha negra)
- Modo de vía (automática, sólo tag, manual, etc)
- Estado de la vía: abierta (representado con flecha verde), cerrada (representado con aspa roja) o pausa (representado con una P).
- Si existe o no vehículo. Si lo hay se representará con el icono de un vehículo.
- Si hay alarma en esa vía. Si la hay se representará con un icono rojo.

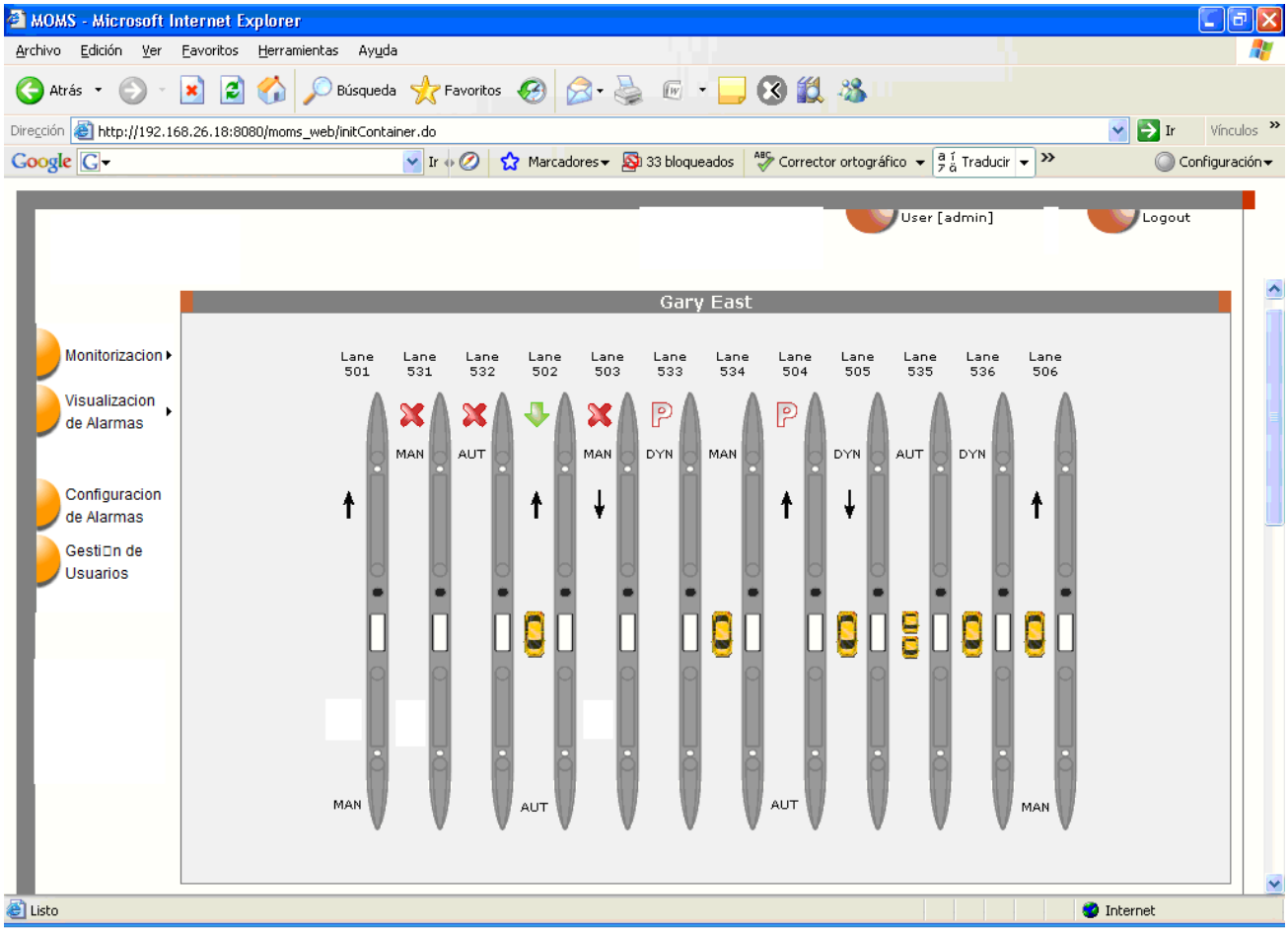

Figura 6: Monitorización de estación

Para acceder a la funcionalidad de Monitorización de Estación, será necesario que el usuario tenga el rol de Controlador de Estación o de Administrador.

8.2.3 VISUALIZACIÓN DE ALARMAS EN TIEMPO REAL

Esta funcionalidad permite tener un listado en tiempo real de todas las alarmas activas en el peaje. Esta visualización tendrá posibilidades de filtrado por los siguientes campos:

- Lugar donde se ha producido la alarma
- Tipo de alarma generada
- Elemento que generó la alarma
- Proceso que generó la alarma o que controla al dispositivo que tiene alarma

Además, existe la posibilidad de ordenar la lista mediante los siguientes criterios:

- Fecha Hora en que se produjo la alarma
- Elemento que generó la alarma
- Lugar donde se ha producido la alarma
- Tipo de alarma generada

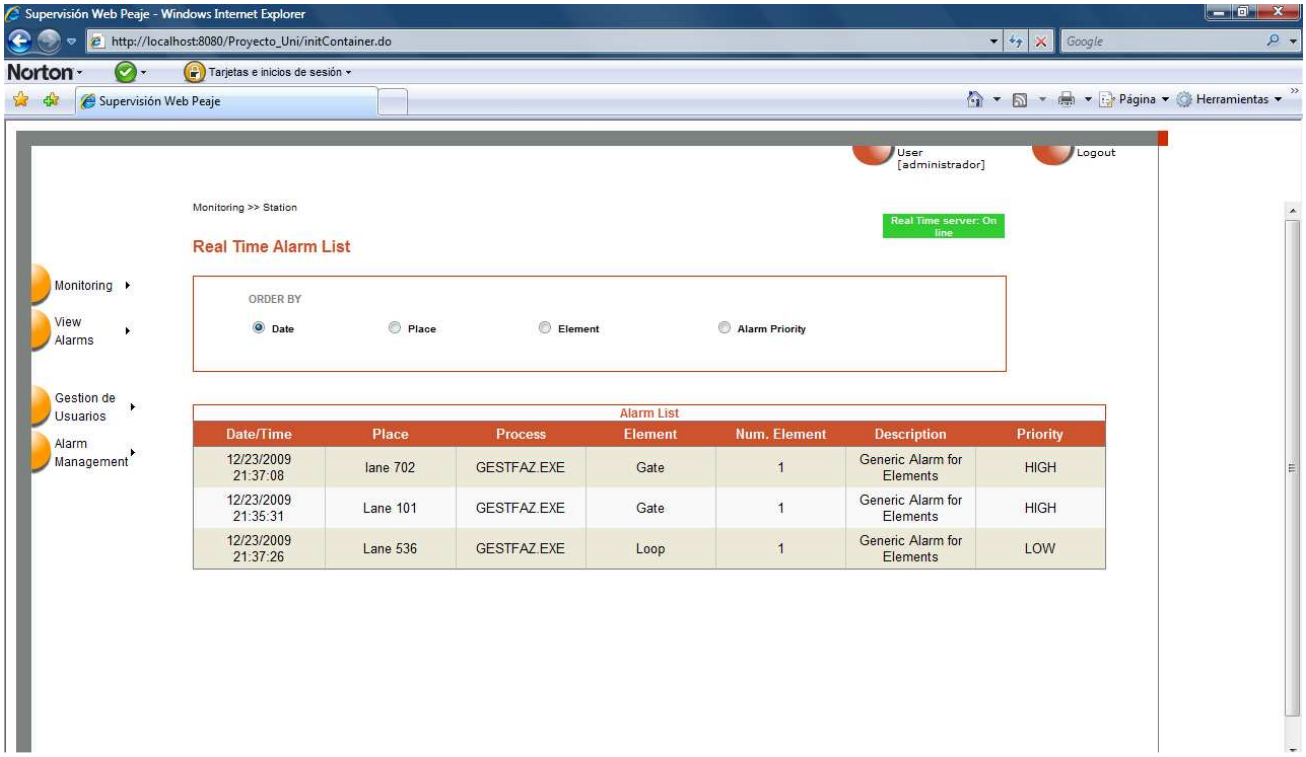

Figura 7: Monitorización de Alarmas

La información que se mostrará de cada alarma generada, será:

- Fecha/hora en que se generó a alarma.
- Proceso que generó la alarma o que controla al dispositivo que tiene alarma.
- Elemento que tiene alarma
- Código Alarma. Indica el tipo de alarma producida
- Prioridad de la alarma
- Lugar del peaje donde se ha producido la alarma

Para acceder a la funcionalidad de Monitorización de Estación, será necesario que el usuario tenga el rol de Visualización de Alarmas o de Administrador.

8.2.4 VISUALIZACIÓN DE HISTÓRICOS DE ALARMAS

En este módulo es posible realizar una búsqueda en Base de Datos de Históricos de las alarmas almacenadas. Como se ha dicho anteriormente, a medida que las alarmas se van generando, existen procesos encargados de recogerlas y de ir insertándolas en Base de Datos para así poder tener históricos con los que poder realizar estudios y análisis posteriores.

La búsqueda se puede realizar mediante el siguiente filtro:

- Fecha / Hora Desde y Hasta. Estos dos parámetros son obligatorios debido a la gran cantidad de registros que pueden resultar de una búsqueda si este filtro.
- Lugar de la alarma
- Proceso que generó la alarma
- Evento que puede ser de inicio o de fin de alarma
- Tipo de Alarma generada
- Prioridad de la alarma
- Elemento que produjo la alarma
- Número de elemento que produjo la alarma

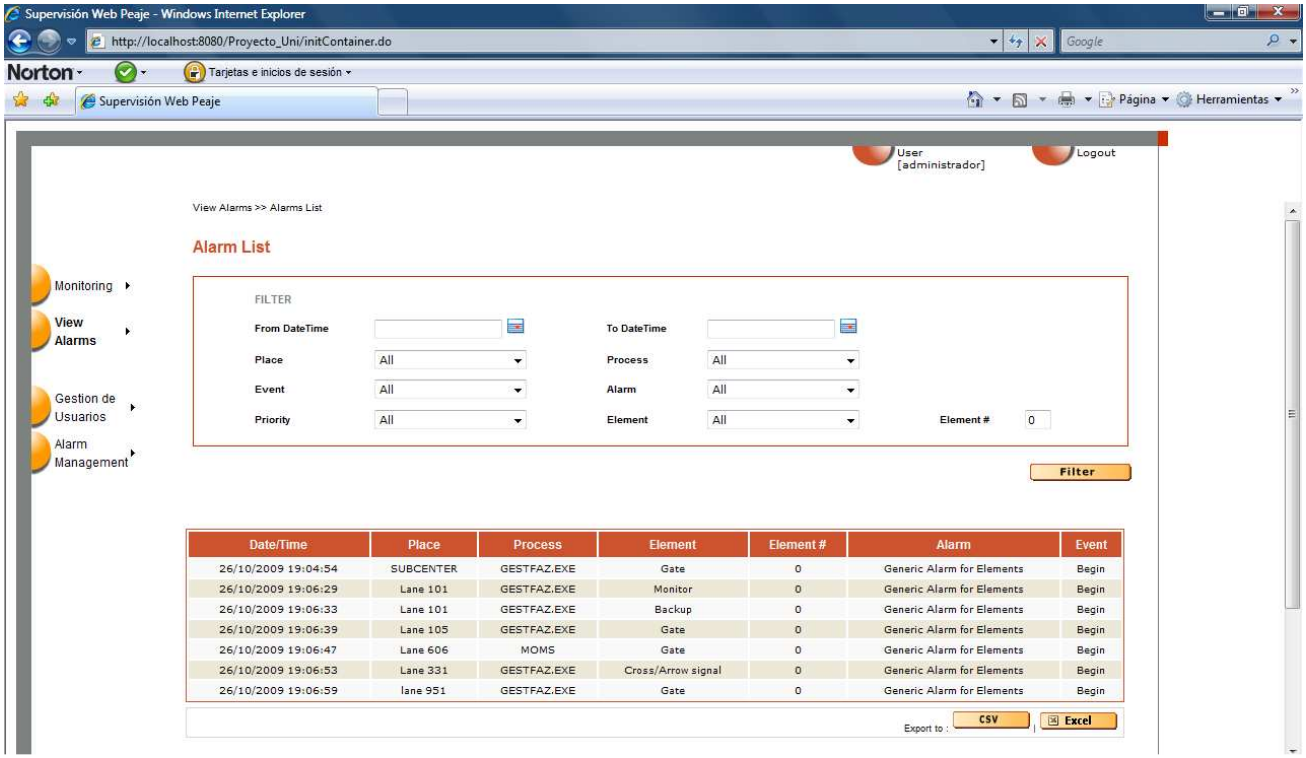

Figura 8: Históricos de alarmas

La información mostrada de cada alarma será la siguiente:

- Fecha/hora en que se generó a alarma.
- Proceso que generó la alarma o que controla al dispositivo que tiene alarma.
- Elemento que tiene alarma
- Código Alarma. Indica el tipo de alarma producida
- Prioridad de la alarma
- Lugar del peaje donde se ha producido la alarma

Además, el listado podrá ser exportado a CSV para su posterior visualización y análisis.

Para acceder a la funcionalidad de Monitorización de Estación, será necesario que el usuario tenga el rol de Visualización de Alarmas o de Administrador.

8.3.MÓDULO DE GESTIÓN Y CONFIGURACIÓN

Este módulo engloba fundamentalmente dos funciones de principales de configuración del sistema. El primero es la configuración de la prioridad de la prioridad de las alarmas y la gestión de usuarios.

8.3.1 CONFIGURACIÓN DE ALARMAS

En este apartado se permitirá configurar la prioridad que tendrá cada tipo de alarma. En función de la prioridad de la alarma, se desencadenarán las acciones pertinentes para tratarla.

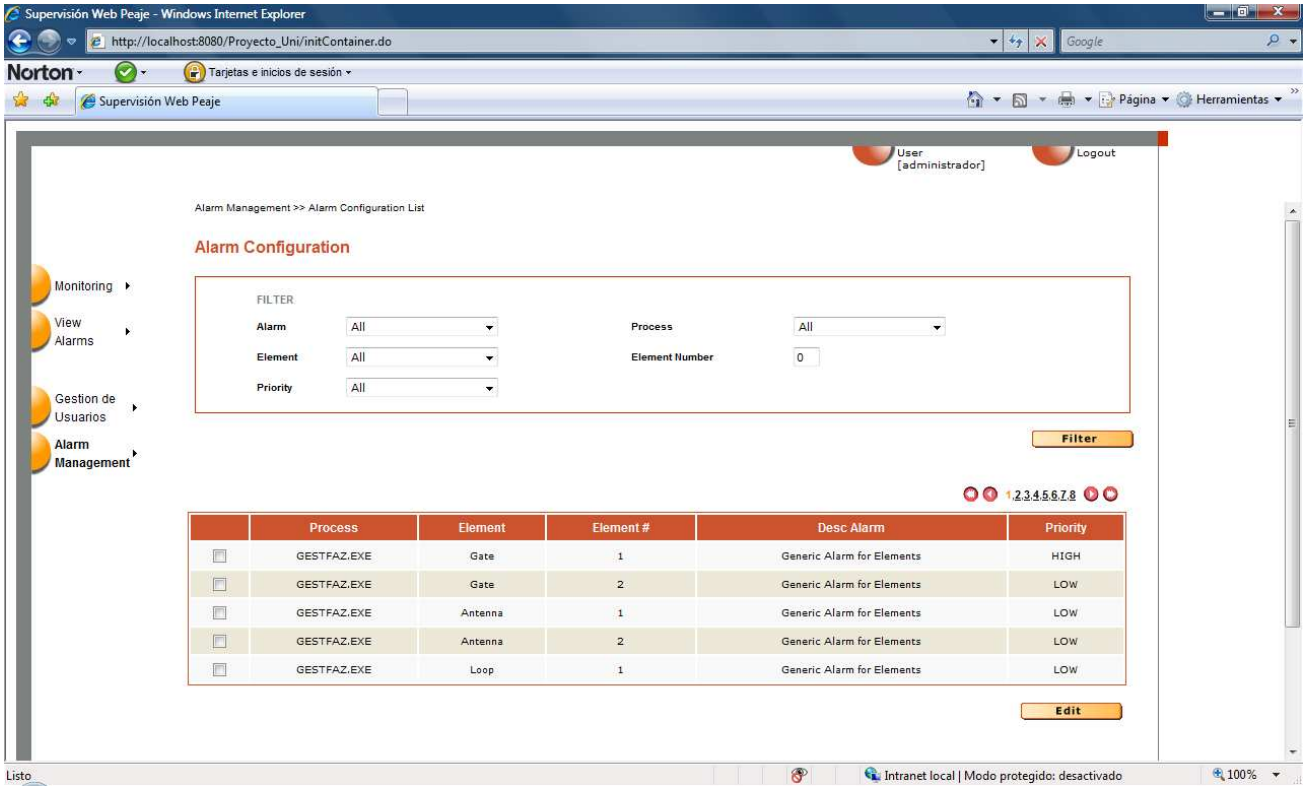

Figura 9: Configuración de alarmas

Un filtro permite listar el conjunto completo de alarmas que pueden darse en el sistema. Editando el detalle de una alarma en cuestión, se pasa a la página de detalle de alarma donde se puede elegir la prioridad de esa alarma. También permitirá configurar la prioridad de la alarma.

Para acceder a la funcionalidad de Monitorización de Estación, será necesario que el usuario tenga el rol de Configuración o de Administrador.

8.3.2 GESTIÓN DE USUARIOS

Mediante el Módulo de Gestión de Usuarios se permite a los usuarios habilitados, la asignación de roles predefinidos a usuarios del sistema.

Los roles definidos en el sistema ya se han detallado anteriormente y se repiten aquí por venir a colación:

- **Administrador**: permite el acceso a todas las partes de la aplicación.
- **Controlador de Estación**: permite monitorizar las estaciones del peaje.
- **Controlador de Centro**: permite monitorizar el Centro de Control.
- **Visualización de Alarmas:** permite visualizar alarmas en tiempo real e históricos.
- **Configuración:** permite acceder el módulo de administración y configuración.

La Figura 28 muestra el modo en que se asocian los roles a los usuarios.

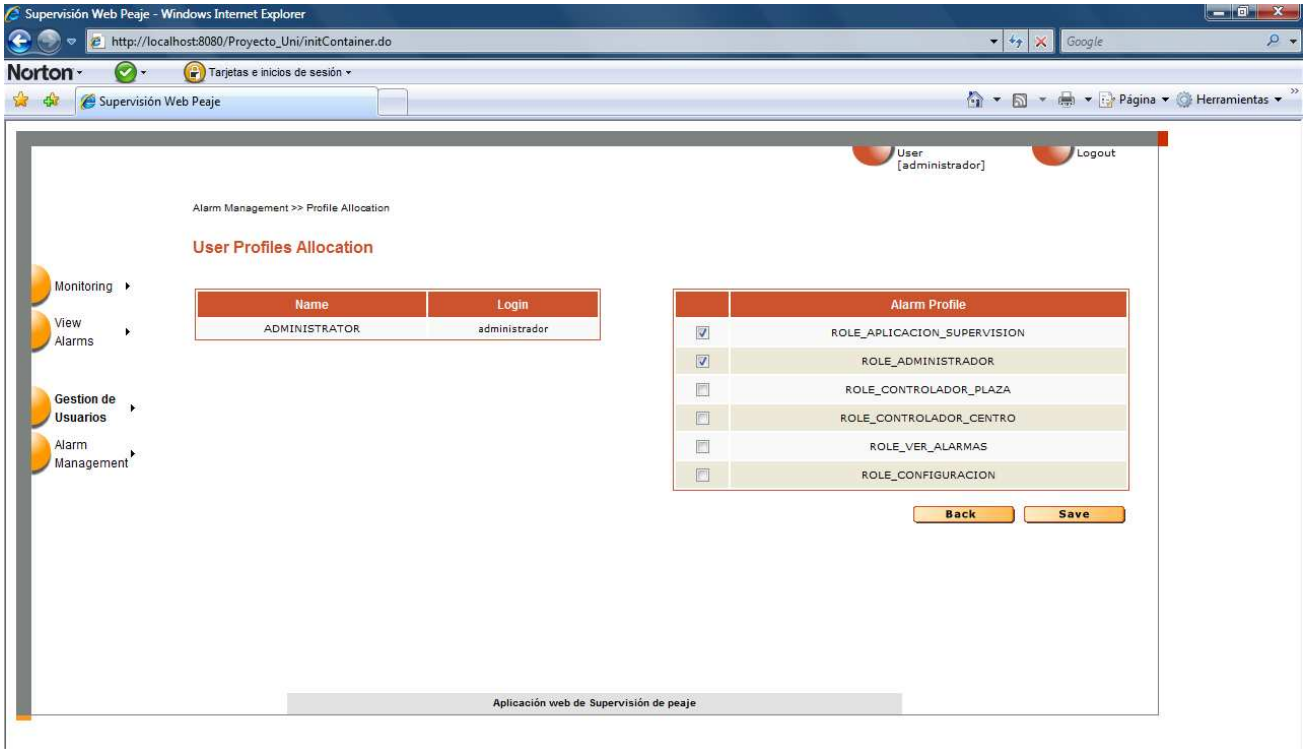

Figura 10: Configuración de usuarios

Para acceder a la funcionalidad de Monitorización de Estación, será necesario que el usuario tenga el rol de Configuración o de Administrador.

9 DESARROLLO DE LA APLICACIÓN

En el apartado dedicado a la Descripción Técnica de la aplicación, se enumeraban los frameworks empleados en el desarrollo de la aplicación y se daba una descripción de cada uno de ellos. Este apartado completa aquel concretando el modo en que estos frameworks han sido utilizados, cómo han sido configurados y qué utilidad exactamente se les ha dado en el conjunto de la arquitectura.

En definitiva, en este apartado se van a detallar los aspectos más importantes acerca del desarrollo de la aplicación extrayendo algunos fragmentos de código en los casos que se considere necesario para ilustrar la explicación. El objetivo es aportar una visión técnica de cómo se han implementado ciertas funcionalidades.

9.1. HIBERNATE

Lo primero que se va a describir es la capa de persistencia puesto que constituye los cimientos de la aplicación. Hibernate es el framework que nos permite acceder a las Bases de Datos y realizar operaciones sobre las tablas pudiendo utilizar tanto el lenguaje **HQL (Hibernate Query Language)** como utilizar sentencias en SQL nativo.

HQL pone a nuestra disposición una amplia gama de métodos que nos permite, además de realizar las operaciones típicas de SQL como INSERTS, DELETES, UPDATES, etc… otras muchas operaciones refiriéndonos siempre a objetos y propiedades Java, lo cual simplifica en gran medida el desarrollo. El truco está en definir un mapeo previo que asigna tablas a objetos Java y propiedades a campos.

Los objetos Java básicos sobre los que se proyectan las tablas, se llaman **POJO's** (Java Objects), mientras que las clase donde se definen los métodos que acceden a las Bases de Datos se llaman **DAO's** (Data Access Objects).

Los mapeos entre los objetos Java y las tablas se definen en unos ficheros XML con extensión **'.hbm.xml'**. Es fundamental un correcto diseño de la Base de Datos para un mapeo coherente y eficaz. Hibernate es completamente intolerante ante las incoherencias en el diseño de la Base de Datos.

Para explicar cómo funciona, nada mejor que el ejemplo de un mapeo que tiene mucha miga, como es el de la tabla H_ALARMA, cuyo significado y diseño ya se ha explicado en el apartado dedicado al Modelo de Datos.

La tabla H_ALARMA, tiene una "Primary Key" compuesta por siete campos. Además tiene seis campos que son "Foreign Keys" de otras tablas. Cinco de estos últimos son parte de la Clave Primaria.

Puesto que la Clave Primaria de la tabla está compuesta por siete campos, es necesario crear dos clases Java: una para definir la clave que llamaremos **"HAlarmaId.java"** y otra para definir el objeto donde se mapea el registro de la tabla en sí que llamaremos **"HAlarma.java"**. A continuación muestro las propiedades de cada una de las clases.

HAlarmaId.java

```
private Date fechaHora;
private Lugares lugares;
private DescProceso descProceso;
private DescAlarma descAlarma;
private DescElemento descElemento;
private DescEventoAlarma descEventoAlarma;
private Long nelemento;
```

Donde las clases Lugares.java, DescProceso.java, DescAlarma.java, DescElemeto.java y DescEventoAlarma.java son POJO's mapeadas con las tablas a las que hacen referencia las correspondientes Claves Foráneas de la PK de la tabla H_ALARMA. Todas estas tablas fueron descritas en el apartado dedicado al Modelo de Datos.

HAlarma.java

```
private HAlarmaId id;

private DescPrioridadAlarma descPrioridadAlarma;
private Long zonaHoraria;
private String info;
private byte procesado;
```

La primera propiedad es la clave primaria de la tabla que hace referencia al objeto **HAlarmaId** definido anteriormente.

Ahora veamos cómo se realiza el mapeo en el fichero **HAlarma.hbm.xml.**

La primera línea indica la ruta a la clase y el nombre de la tabla:

```
<class name="${PATH_HAlarma}" table="H_ALARMA">
```

A continuación se mapea la Clave Primaria:

```
<composite-id name="id" class="${PATH_HAlarmaId}">
    <key-property name="fechaHora" type="java.util.Date" column="FECHA_HORA" />
    <key-many-to-one name="lugares" class="${PATH_Lugares}" column="ID_LUGAR" />
    <key-many-to-one name="descProceso" class="${PATH_DescProceso}" column="ID_PROCESO" />
    <key-many-to-one name="descAlarma" class="${PATH_DescAlarma}" column="ID_ALARMA" />
    <key-many-to-one name="descElemento" class="${PATH_DescElemento}" column="ID_ELEMENTO" />
    <key-many-to-one name="descEventoAlarma" class="${PATH_DescEventoAlarma}" column="ID_EVENTO_ALARMA" />
    <key-property name="nelemento" type="java.lang.Long" column="N_ELEMENTO" />
</composite-id>
```

Como se puede observar, todas las propiedades que componen la clave primaria están declaradas dentro de la etiqueta **<composite-id>,** que indica que se trata de una clave compuesta. En la declaración de dicha etiqueta, se especifica el nombre de la propiedad (name=“id”) y la clase java que la define (class=“HAlarmaId”).

Las propiedades que están relacionadas con otro objeto, se definen bajo la etiqueta **<key-many-to-one>,** que indica que es una relación de muchos (en la tabla origen) a uno (en la tabla destino). En la declaración de este TAG, se especifica el nombre de la propiedad (name), la clase java que la define (class) y el campo que se corresponde con la clave primaria en la tabla destino (column).

Las propiedades que no son Clave Foránea de ninguna otra tabla, se definen con la etiqueta **<key-property>,** indicando nuevamente el nombre de la propiedad (name), el tipo de dato (type) y el nombre de la columna (column).

El resto de las propiedades que no componen la Clave Primaria se declaran a continuación utilizando la misma nomenclatura excepto el indicador "key".

Existen varios tipos de mapeo en Hibernate. También existen las etiquetas **one-to-one** y **one-to-many**. Esta última es la inversa del **many-to-one** que he indicado antes y permite obtener un conjunto de valores insertados en una tabla con un identificador que se corresponde con la PK de origen. Hibernate almacena dicho conjunto de registros en algún tipo de lista que proporciona Java tales como Set, List o Map.

Los mapeos en Hibernate se pueden complicar tanto como queramos y nos exija el diseño de la Base de Datos. No es conveniente extenderse más en este tema, para no desviarse del objetivo de este apartado.

9.2.SPRING

Spring es el framework que sirve de eslabón de enganche entre la capa de persistencia y la capa de negocio. Spring pone un conjunto de librerías a nuestra disposición y permite definir en unos ficheros de configuración la manera en que la lógica de negocio va a acceder a la capa de persistencia.

Además proporciona una serie de métodos y templates que nos permiten delegar en él la gestión de las sesiones de persistencia, así como utilizar un conjunto de métodos para que Hibernate realice sobre las Bases de Datos las operaciones deseadas.

En la aplicación web Spring se ha configurado en base a dos ficheros que se describen brevemente a continuación:

- **applicationContext-dao.xml**

En él se configuran los Data Access Objects, es decir, las clases Java que acceden directamente a las Bases de Datos por medio de los métodos que las librerías del framework ponen a nuestra disposición y que nos evitan tener que manejar directamente las sesiones de Hibernate.

En primer lugar se declara el bean **DataSource,** que define el origen de los datos, es decir, todos los parámetros necesarios para conectarnos con la Base de Datos. En nuestro caso estos parámetros los tenemos mapeados en un fichero llamado "database.properties", y de ahí los extraemos.

```
<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location"><value>classpath:/database.properties</value></property>
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName"><value>${driver}</value></property>
    <property name="url"><value>${url}</value></property>
    <property name="username"><value>${username}</value></property>
    <property name="password"><value>${password}</value></property>
    <property name="maxActive"><value>${maxActive}</value></property>
    <property name="maxWait"><value>${maxWait}</value></property>
    <property name="maxIdle"><value>${maxIdle}</value></property>
    <property name="removeAbandoned"><value>${removeAbandoned}</value></property>
    <property name="removeAbandonedTimeout"><value>${removeAbandonedTimeout}</value></property>
    <property name="logAbandoned"><value>${logAbandoned}</value></property>
    <property name="defaultReadOnly"><value>${defaultReadOnly}</value></property>
    <property name="validationQuery"><value>${validationQuery}</value></property>
</bean>
```

A continuación se declaran un objeto de Hibernate llamado **SessionFactory** que maneja las sesiones que ejecutan las distintas transacciones a la Base de Datos. Dentro de esta definición, se declaran todos los ficheros de Hibernate (hbm.xml) que se pretenden mapear.

```
<!-- Hibernate SessionFactory Definition -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>

    <property name="mappingResources">
        <list>
            <value>../hbm.xml</value>

        </list>
    </property>

    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
            <prop key="hibernate.cglib.use_reflection_optimizer">true</prop>
            <prop key="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</prop>
            <prop key="hibernate.cache.use_query_cache">true</prop>
        </props>
    </property>
</bean>
```

También hay que declarar el bean que se encarga del manejo de transacciones, llamado **TransactionManager**.

```
<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

A continuación hay que definir uno por uno todos los beans de persistencia que se van a manejar en la aplicación.

```
<bean id="HAlarmaDAO" class="${PATH_HAlarmaDAOImpl}">
    <property name="sessionFactory"><ref local="sessionFactory" /></property>
</bean>
```

En el DAO definido se implementan todos los métodos de operaciones con la Base de Datos. Para ello disponemos del "Hibernate Template", que nos permite manejar el framework de spring. Para utilizar el Template de hibernate, debemos hacer que el DAO extienda la clase HibernateDaoSupport.

Este Template es la clave de los DAO. Se encarga de guardar, borrar, listar y obtener los objetos de la Base de Datos. También se encarga de manejar las excepciones, lo cual nos ahorra mucho trabajo.

- **applicationContext-service.xml**

Aquí definimos una serie de clases que denominamos Services, que van a ser métodos o puntos de acceso al servicio de persistencia a través de los cuales la capa de negocio va a ejecutar ciertas operaciones sobre las Bases de Datos de forma completamente transparente para ella.

Dentro de la declaración de cada Service, hay que declarar también cuales son los DAOs que se van a utilizar, que pueden ser uno o varios.

```
<bean id="HAlarmaService" class="${PATH_HAlarmaServiceImpl}" singleton="false">
    <property name="HAlarmaDAO"><ref bean="HAlarmaDAO" /></property>
</bean>
```

Se podría decir que con Spring hemos configurado de manera limpia, fácil, transparente y completamente modular un interfaz entre las capas de negocio y de persistencia.

Cada vez que queramos implementar una nueva operación sobre las Bases de Datos tendremos, en primer lugar que definirla en el DAO sirviéndonos de las herramientas disponibles para realizar cualquier tipo de operación sobre las mismas. Una o una combinación de estas operaciones así como cualquier procesado anterior o posterior que se quiera hacer de los datos se puede implementar a nivel de Service para que quede a disposición de la capa de negocio.

9.3.STRUTS

En el framework de Struts XML define la lógica de presentación y navegación de la aplicación. Se configura en uno o varios ficheros struts-config.xml. En este fichero van registrados:

- **Beans de formulario**, que son los datos que envía cada acción del usuario al controlador.

```
<form-beans>
    <form-bean name="HAlarmaForm" type="${PATH_HAlarmaForm}"/>
</form-beans>
```

- Acciones de formulario, que son las acciones que pueden ser llamadas desde las páginas del cliente. Cada acción recibe información mediante los beans del formulario y otros beans definidos en la aplicación y responde al usuario generando diferentes vistas mediante JSP. Es en las acciones donde se escribe la lógica del negocio y la gestión del modelo de datos y su persistencia.

- Reglas de navegación por las diferentes páginas, mediante los mapeos de acciones (**action mappings**)

```
<action-mappings>

    <action attribute="HAlarmaForm" name="HAlarmaForm"
        path="/Alarma/AlarmaList"
        type="${PATH_HAlarmaListAction}"
        scope="request"
        validate="false">
        <forward name="success" path="listAlarma" />
    </action>

 </action-mappings>
```

- Reglas de validación de datos de los formularios, que permiten tanto validación en cliente como en el servidor bajo el control de la clase **ValidatorPlugIn** del framework de Struts.

```
<form name="/Alarms/AlarmsInsert">

    <field property="alarms.description" depends="required, mask">
        <arg0 key="Message.profile.description"/>
        <arg1 key="Message.invalid.AlphaNumeric" />
        <var>
            <var-name>mask</var-name>
            <var-value>^[0-9a-zA-ZáéíóúüÁÉÍÓÚÜ ñ.,-]*$</var-value>
        </var>
    </field>

</form>
```

- Tiles utilizados para hacer el forward de las acciones bajo el control de la clase **TilesPlugin** del framework de Struts.

```
<!-- CONFIGURACION DE LISTAS DE ALARMAS -->
<definition name="listAlarma" extends="layout">
    <put name="validation" value="/WEB-INF/jsp/utils/blank.jsp"/>
    <put name="navigation" value="Alarms --> List of Alarms" type="string"/>
    <put name="tittle" value="List of Alarms" type="string"/>
    <put name="body" value="/WEB-INF/jsp/pages/alarmas/listAlarma.jsp"/>
</definition>
```

- Etiquetas para internacionalización declaradas bajo la etiqueta **<message-resources>**

9.4.MONITORIZACIÓN DE ALARMAS EN TIEMPO REAL

La aplicación tiene varias funcionalidades que requieren mostrar en tiempo real la información disponible. Concretamente, los módulos que muestran información en tiempo real son los siguientes: Monitorización de Centro, Monitorización de Estación y Monitorización de Alarmas. Ambas tres funcionalidades están descritas en el apartado dedicado a la Descripción Funcional.

En este apartado se describe el flujo de acciones y datos que se produce durante la monitorización en tiempo real. Para ello se va a exponer el módulo de monitorización de alarmas. El funcionamiento es el mismo en los casos de monitorización de Centro y de Estación. Lo que cambia es la información obtenida y la manera de representar esta información en el interfaz web.

A continuación se detalla el proceso de refresco en tiempo real de las páginas de monitorización. Sintetizando, los pasos se repiten periódicamente y son los siguientes:

1. El cliente invoca al método servidor
2. El servidor realiza la petición al Servidor en Tiempo Real y recibe las información requerida
3. Se parsea, procesa y prepara la información recibida y se envía al cliente
4. El cliente construye la tabla y la muestra en el interfaz al usuario
5. Se repite el proceso cada cierto periodo de refresco

A continuación se detalla paso a paso este proceso.

- **Configuración**

En primer lugar vamos a ver qué se le dice a DWR (Direct Web Remoting) en el fichero de configuración **dwr.xml:**

```
<dwr>
    <allow>
        <convert converter="bean" match="${PATH_BEANS}.*"/>
        <create creator="spring" javascript="realTimeWService">
            <param name="beanName" value="RealTimeWService"/>
        </create>
        <convert converter="exception" match="java.lang.Exception"/>
        <convert converter="bean" match="java.lang.StackTraceElement"/>
    </allow>
</dwr>
```

Donde le damos la ruta a los beans para los que queremos una conversión entre Java y javascript (converter=”bean”). Además le decimos que queremos integración con Spring y le decimos el servicio que va a hacer de puente entre la parte cliente y la parte servidor. En nuestro caso tenemos los servicios escritos en javascript en **“realTimeWService.js”**, mientras que los servicios en la parte servidor están en **“RealTimeWService.java”.**

De esta manera, invocaremos dichos servicios desde la parte cliente tal y como si estuviéramos en la parte servidor. Cada método en la parte cliente invoca algún método en la parte servidor. Las librerías de dwr se encargarán posteriormente de realizar las conversiones de datos que sean necesarias.

Además estamos declarando un manejador de excepciones en dwr, de modo que cuando se produzca una excepción en el código del servidor, seamos capaces de capturarla desde el lado cliente para actuar en consecuencia.

- **refrescoAlarmas.jsp**

El funcionamiento de la jsp que muestra al usuario las alarmas en tiempo real que se van sucediendo en el sistema, se resume en las siguientes líneas de código javascript:

```
//Funcion de que se ejecuta cada periodo configurable
function callGetTodasAlarmas(){

      realTimeWService.getTodasAlarmas( {

            callback:refrescoAlarmas,
            timeout:intervaloTimeOut,
            errorHandler:function(javaClassName) {

                alert(GENERIC_SERVER_ERROR);

                refrescarTablaAlarmas(new Array(),"listado_alarmas",columnHeadersAlarms);


                  }
            });

      }

//Funcion que se encarga de refrescar los resultados de la tabla
function refrescoAlarmas(listaalarmas) {

                  ..................
            refrescarTablaAlarmas(listaalarmas, "listado_alarmas", columnasTablaAlarmas);
                  ..................

      }

//Indicamos el periodo de refresco y la función que debe ejecutarse
setInterval(callGetTodasAlarmas, periodoRefresco);
```

La última línea de código (**setInterval(callGetTodasAlarmas, periodoRefresco**)), lleva como primer parámetro el nombre de la función que se debe ejecutar periódicamente, mientras que el segundo parámetro es el periodo de refresco.

La función **callGetTodasAlarmas()** dice lo siguiente:

1. Que se ejecute la función **realTimeWService.getTodasAlarmas()** que está definida en el fichero **realTimeWService.js** y a su vez tiene un servicio paralelo en el servidor.

   Como se puede observar, invocamos al servicio de la misma manera que si estuviéramos en el lado servidor.

2. Decimos también qué función queremos que se ejecute cuando se obtenga respuesta por parte del servidor. Es lo que definimos como **callback**. En este caso, invocamos a la función **refrescoAlarmas(listaAlarmas)** que acepta como parámetro la lista obtenida tras la consulta y ya convenientemente convertida a objetos javascript por el **"engine"** de dwr.

   Como se puede observar, el funcionamiento es completamente asíncrono. Es decir, mientras el servidor hace las operaciones necesarias para obtener la información, el script del cliente sigue su ejecución. No es bloqueante, sino que tiene un punto de retorno en la función definida como callback.

3. Definimos el **timeout**, intervalo fuera del cual dwr desliga el punto de retorno de la petición y realiza una petición nueva.
4. Definimos también qué hacer en caso de que el manejador de Excepciones (**errorHandler**) detecte que se ha producido una excepción que puede ser tanto en servidor como en cliente.

- **Lado del Cliente: realTimeWService.getTodasAlarmas()**

La función en el lado cliente que invoca al método Java en el servidor es muy sencilla:

```
function getTodasAlarmas(){

      //llamada asincrona
      dwr.engine.setAsync(true);

    // Llamada a método Java en el servidor
      var res = realTimeWService.getTodasAlarmas();

    return res;
}
```

1. Se le indica que la llamada sea asíncrona
2. Se invoca el método implementado en el servidor
3. Se devuelve la respuesta ya convertida a objetos javascript

- **Lado del Servidor: RealTimeWService.getTodasAlarmas()**

En el lado servidor el proceso es el siguiente:

1. Realizar la petición al Servidor en Tiempo Real al puerto HTTP configurado al efecto. Concretamente se le hace una petición extendida al Servidor en Tiempo Real del tipo:

   http://DIR_IP : PUERTO/alarmas

2. El Servidor en Tiempo Real responde con todas la alarmas que tiene almacenadas del sistema encapsuladas en un mensaje regido según el esquema MensajeWeb.xml.

3. Dicho mensaje se lee byte a byte mediante un objeto **InputStream** y se parsea.

A continuación se construyen objetos fácilmente manejables por el código de cliente completando información descriptiva mediante acceso a las Bases de Datos. Dichos objetos se procesan y se ordenan convenientemente para ser enviados al cliente.

Es fundamental una buena configuración del sistema para que no existan incoherencias entre la información recibida en tiempo real y la información almacenada en las tablas de configuración.

4. La información se envía al navegador y este se encarga de mostrarla en el interfaz mediante la ejecución de código javascript.

- **Refresco de la tabla en el interfaz gráfico**

Una vez recibidos los datos del servidor, la función **refrescarTablaAlarmas(lista, div, columnas)**, se encarga de pintar la tabla de alarmas con ayuda de las librerías javascript de Yahoo User Interface.

```
function refrescarTablaAlarmas(listaAlarmas, idDiv, columnas) {


      var columnSetAlarms = new YAHOO.widget.ColumnSet(columns);

      // convertimos lista en array

      // construimos objeto con el contenido de la tabla
      dataSourceAlarms = new YAHOO.util.DataSource(listaAlarmas);
      dataSourceAlarms.responseType = YAHOO.util.DataSource.TYPE_JSARRAY;
      dataSourceAlarms.responseSchema = {
            fields: [

                  columns[0].key, ……,      columns[N].key

                    ]
            };

// Generamos la tabla
dataTableAlarms = new YAHOO.widget.DataTable(idDiv, columnSetAlarms, dataSourceAlarms);

}// refrescarTablaAlarmas
```

La función admite tres parámetros de entrada: la lista obtenida, el tag div de la página html donde se quiere mostrar la tabla y un array con las leyendas de las columnas.

Para la construcción de la tabla, utilizamos los constructores y objetos proporcionados por las librerías javascript de YUI:

- Objeto de leyendas de las columnas: **new YAHOO.widget.ColumnSet(columnas)**
- Indicamos el tipo de origen de los datos: **YAHOO.util.DataSource.TYPE_JSARRAY**
- Origen de los datos: **new YAHOO.util.DataSource(listaAlarmas)**
- Generamos la tabla y la pintamos:

  new YAHOO.widget.DataTable(idDiv, columnSetAlarms, dataSourceAlarms)
- YUI aporta unos CSS para definir los estilos de las tablas

9.5.HISTÓRICOS DE ALARMAS

En este punto se explica cómo se ha implementado el acceso a base de datos utilizando **HQL (Hibernate Query Language)** a partir de la funcionalidad de consulta de los **históricos de alarmas**.

Este ejemplo aporta una explicación que puede resultar muy ilustrativa en este aspecto debido a que entraña una extracción de los datos limitada por el filtrado de búsqueda escogido por el usuario. El filtrado puede ser determinado por un parámetro, ninguno o una combinación de varios.

Lo primero es diseñar un formulario (**FormBean**) adecuado tanto para recoger los datos del filtro seleccionado por el usuario como para devolver los resultados de la búsqueda. Dichos formularios y el flujo de acciones de navegación se controladas por el framework de Struts.

En nuestro caso tenemos un FormBean llamado **HAlarmaForm.java** diseñado para servir de puente entre el interfaz de usuario y la capa de negocio. Recoge los datos seleccionados por el usuario en el interfaz y le devuelve los datos obtenidos de la búsqueda.

De este modo, **HAlarmaForm** tiene algunas propiedades dedicadas a almacenar la información que se muestra en los combos de filtrado.

```
//Listado de combos.
    private List<Lugares> listaLugares;
    private List<DescProceso> listaDescProceso;
    private List<DescElemento> listaDescElemento;
    private List<DescAlarma> listaDescAlarma;
```

Otro para almacenar los criterios de búsqueda seleccionados:

```
//Atributo para almacenar los criterios de busqueda
    private HAlarma halarma;
```

Y otra lista para almacenar los resultados que se le devuelven al cliente

```
//Lista de resultados
    private List<HAlarma> listaAlarmas;
```

A partir de ahí, el controlador de Struts se encarga de enviar el formulario al Action encargado de realizar las operaciones necesarias para realizar la búsqueda y devolver los resultados obtenidos.

Dicho Action, se apoya en la capa de persistencia por medio del servicio configurado a tal efecto y el Data Access Object (DAO) correspondiente ejecuta la búsqueda tal y como se describe a continuación.

```
public List<HAlarma> getAllHAlarma(HAlarma hAlarma) throws DAOException {

	List<HAlarma> list = null;
	Criteria criteria = null;
	Session session = null;

	try {

		//Obtenemos la sesion
		session  = this.getHibernateTemplate().getSession();

		// Creamos el criteria para el filtrado de las alarmas
		criteria = session.createCriteria(HAlarma.class);

		// Rellenemos el criteria con los valores que vengan en el objeto
		fillListCriteria(criteria, HAlarma);

		//Listamos los resultados
		list = criteria.list();

	} catch (Exception e) {

		LogUtils.showError(e, logger, "Error al obtener alarmas");
		throw new DAOException(e.getMessage(), getCause(e));

		} finally {
			if (session != null) {
				session.close();
			}
		}

		//Devolvemos el resultado
		return list;

	}
```

1. En primer lugar obtenemos la sesión que se encargará de ejecutar la búsqueda.

2. Creamos en la sesión un objeto de tipo **Criteria** que almacena los criterios de búsqueda.

   Para construir el objeto "criteria", se van añadiendo uno a uno todos los criterios de búsqueda que se quieran aplicar al listado mediante métodos estáticos del objeto **Expression** tales como **Expression.le** (less or equal), **Expression.ge** (greater or equal), **Expression.eq** (equal), **Expression.between** (between)….

A estos métodos se les pasa dos parámetros: un alias a la propiedad a la que se refieren y el parámetro elegido en cuestión. A continuación se muestra algún ejemplo:

```
//Fecha entre dos dadas
criteria.add(Expression.between("fechaHora", fechaDesde, fechaHasta));

//Fecha Hasta una dada
criteria.add(Expression.le("fechaHora", fechaHasta));

//Fecha a partir de una dada
criteria.add(Expression.ge("fechaHora", fechaDesde));

//idElemento igual a
criteria.add(Expression.eq"descElemento.id", criterioIdElemento);
```

3. Por último se hace el listado. El manejador de la sesión de persistencia se encarga de realizar la búsqueda ateniéndose a los criterios definidos.

4. Se captura la posible excepción que pueda arrojar la ejecución de la transacción

5. Se devuelve el resultado a la capa de negocio para que procese los resultados, los encapsule en el formulario y lo devuelva al cliente para que este se encargue de mostrárselo al usuario mediante s su interfaz.

9.6.GOOGLE MAPS

La funcionalidad de Monitorización de Centro de Control se ha implementado de forma gráfica sobre el mapa de la autopista. Dicho mapa está construido haciendo uso de los métodos proporcionados por el API de Google Maps.

En el apartado dedicado a la Descripción Funcional se han expuesto las distintas funcionalidades que se han desarrollado en este módulo. En este apartado se va a describir el código desarrollado para implementar tales funciones, el cual está escrito íntegramente en javascript.

Antes de nada, hay que apuntar que este módulo lleva implícita una tarea de información en tiempo real de ciertos parámetros del sistema tales como alarmas o número de vías abiertas en cada modo de operación o si están abiertas de entrada o de salida. En este apartado no se va a explicar cómo se extrae y refresca esa información puesto que eso ya se ha explicado en el apartado de Monitorización de Alarmas en Tiempo Real. En este apartado se expone únicamente el modo en que se construyen los objetos en los que se ubica esta información.

Como ya se ha apuntado, el peaje se divide en dos fase debido a su extensión que llamaremos Fase I y Fase II. Para construir el mapa lo primero que se ha hecho es localizar las coordenadas de cada una de las estaciones del peaje. A partir de esa información se han construido 2 ficheros XML estáticos con la información de todos los puntos extraídos. Cada entrada tiene 3 propiedades: **latitud**, **longitud**, **html** de la ventana, **etiqueta** y **categoría**.

```
<markers>

    <marker lat="41.68803233297291" lng="-87.52206802368164" html=""  label="Westpoint" category="fase1" />
    <marker lat="41.64130512983454" lng="-87.50573337078094" html="" label="Calumet Entry" category="fase1" />
    <marker lat="41.63769698918256" lng="-87.50648438930511" html=""  label="Calumet Exit" category="fase1" />
    <marker lat="41.61329263861522" lng="-87.42194652557373" html=""  label="Cline Avenue" category="fase1" />
    <marker lat="41.591972377913706" lng="-87.30491101741791" html=""  label="Gary East" category="fase1" />
    <marker lat="41.59311578793709" lng="-87.23737835884094" html=""  label="Lake Station" category="fase1" />
    <marker lat="41.58017200600028" lng="-87.17302680015564" html=""  label="Willow Creek Road" category="fase1" />

 </markers>
```

Una vez construido el mapa con el constructor GMap2, se lee este fichero extrayendo cada una de las propiedades de que se compone cada entrada y sobre el objeto “mapa” se le van añadiendo los marcadores en cada una de sus estaciones utilizando el método crearMarcador():

```
function crearMarcador (punto,nombre,html,categoria) {

   //Creamos el marcador
   var marcador = new GMarker(punto);

     //ALmacenamos categoria y le ponemos nombre
   marcador.categoria = categoria;
   marcador.nombre = nombre;

   GEvent.addListener(marcador, "click", function() {
       marcador.openInfoWindowHtml(html);
   });

   // guardamos array de marcadores para sidebar
   arrayMarcadores.push(marcador);

   // una linea mas en la barra lateral
side_bar_html += '<a href="javascript:detallePlaza('+(arrayMarcadores.length-
1)+')">' + nombre + '<\/a><br>';

   return marcador;
}
```

1. Creamos marcador con el constructor GMarker
2. Almacenamos nombre y categoría
3. Definimos un manejador del evento “onClick” para ese marcador
4. Guardamos el marcador en un array porque luego vamos a manejar ese array para mostrar u ocultar estaciones en función de la selección del checkbox o lo que es lo mismo, de su categoría.
5. Vamos construyendo las líneas que se mostrarán en la barra lateral. La función javascript definida (**detalleEstacion**) nos dirigirá al detalle de la estación en cuestión.

Por otra parte el control de los checkbox consiste en recorrer el array de marcadores y mostrar u ocultar en cada caso dependiendo de la categoría seleccionada o deseleccionada.

En cada ciclo de refresco se construye un objeto String html con la información capturada y mediante el método **.innerHTML** de javascript, se escribe la nueva información capturada. Lo mismo se realiza con la información de las alarmas capturadas de la estación. Además, en función de la existencia o no de alarma en la estación, se cambia el **.src** del marcador con el objetivo de que este se ponga en rojo cuando hay alarma y permanezca en verde mientras no la hay.

10 PUESTA EN MARCHA DE LA APLICACIÓN

10.1 IMPLANTACIÓN Y FUNCIONAMIENTO EN UN ESCENARIO REAL

Como se ha explicado a lo largo de este documento, a la aplicación de Supervisión subyace una arquitectura compleja de máquinas interconectadas entre sí. En cada una de las máquinas que componen el sistema, existen numerosos procesos ejecutando distintas tareas. Las vías son máquinas en las que se ejecutan procesos que controlan sus dispositivos y generan mensajes de información de los mismos. Las estaciones son también máquinas que controlan una serie de vías en las que también residen procesos que están ejecutando ciertas tareas. Por último, en el Centro de Control reside el servidor web que aloja la aplicación de Supervisión y a la que llega información de todo el peaje.

En principio los requisitos de hardware y software en cada equipo los imponen los procesos que se ejecutan en el mismo. Por lo general, todos los equipos contienen un SO Windows NT puesto que proporciona un entorno satisfactorio para la configuración de la red, de los dominios y cuentas de usuarios, así como para establecer las comunicaciones entre las distintas máquinas con la seguridad requerida.

En cuanto a las Bases de Datos, se suele utilizar SQL Server tanto para las bases de datos locales como para la de Supervisión. Se trata de un motor de Base de Datos ampliamente utilizado y probado por nuestros sistemas y que responde satisfactoriamente a nuestras necesidades. No obstante, el propio diseño abierto de la arquitectura, permite que se pueda utilizar cualquier motor de Base de Datos.

En cuanto a los requisitos hardware, dependen obviamente de la envergadura del sistema de peaje. En general el elemento más sensible lógicamente son las tarjetas de red. Se suelen usar varias tarjetas multipuerto para conmutar entre ellas en caso de fallos. También la arquitectura de las máquinas suele estar provista de doble nodo para conmutar entre ambos en caso de caída de uno de ellos.

Un peaje de una envergadura media puede tener en torno a cien vías funcionando. Esto quiere decir que tenemos cien vías generando mensajes y, por lo tanto, un volumen de tráfico de mensajes enormemente grande. Esto implica que para el correcto funcionamiento del sistema se tiene que contar con un amplio ancho de banda en las comunicaciones que deberá ser mayor cuanto mayor sea el peaje. En general se utilizan comunicaciones por fibra óptica. Además se hacen pasar los mensajes por

diferentes niveles con el objetivo de ir escalando el envío de información y reducir de este modo los riesgos de congestión en la red.

10.2 VALORACIÓN ECONÓMICA

Tal y como se ha destacado en el apartado introductorio, el objetivo final de este proyecto es dotar a la empresa concesionaria de la explotación de un peaje de carreteras de una herramienta de supervisión del sistema cuyo objetivo último es reducir costes y, en consecuencia, aumentar los beneficios de dicha explotación.

El ahorro en costes y el aumento de los beneficios de la explotación como consecuencia de la utilización de la herramienta web de Supervisión viene motivado por tres causas principales:

- Lamentablemente, la primera posibilidad de ahorro que brinda esta aplicación recae en la reducción de personal.

  Se hace prescindible en torno al 75% de controladores de estación, puesto que el trabajo que hacían antes controlando in situ en cada estación las posibles incidencias que se podían producir, resulta posible hacerlo ahora desde el cómodo puesto del Centro de Control donde tan sólo dos controladores monitorizan el peaje 24 horas al día.

  Debido a la gran extensión de la autopista, era necesario tener a un gran número de empleados de mantenimiento cubriendo ciertas demarcaciones geográficas para tratar de reducir lo máximo posible el impacto que pudiera tener el alto tiempo de respuesta empleado para reparar averías y fallos.

  Con una aplicación de mantenimiento capaz de procesar de forma eficiente e incluso de forma automática, si así se configura, las alarmas generadas por el sistema de Supervisión, los empleados de mantenimiento pueden consultar en tiempo real en sus PDA las incidencias que tienen asignadas.

  La disminución en el tiempo de respuesta ante incidencias y una gestión centralizada de la generación y posterior asignación de partes de trabajo ante incidencias puede suponer una

reducción de hasta un 50% en el personal de mantenimiento, puesto que se necesita menos gente para cubrir las mismas demarcaciones geográficas.

- En segundo lugar, cabe destacar que la reducción en el tiempo de respuesta ante incidencias a la que se aludía anteriormente permite, como es obvio, reducir al mismo tiempo el período de inactividad de las vías y, por tanto, permite ofrecer un mejor servicio a los clientes y aumentar los beneficios hasta en un 20%.

- Existe también un cauce de ahorro muy importante proporcionado por esta aplicación derivado de una mejor previsión ante futuros fallos que se puedan producir en el sistema. A esto se le llama mantenimiento predictivo y consiste en establecer protocolos de mantenimiento para prevenir fallos a partir del análisis de errores producidos previamente en el sistema y consisten generalmente en el cambio o reparación de dispositivos o un control más exhaustivo sobre estos.

- Además de todos estos cauces de ahorro, existe uno que resulta incuantificable pero que no por ello es menos importante, y que resulta de las satisfacción recibida por los clientes al recibir un mejor servicio en la autopista.

  Debido a la mayor celeridad con que se resuelven las incidencias, se causan menos molestias a los clientes, se disminuye el tiempo de inactividad en las vías con fallos, e incluso muchos de ellos logran prevenirse.

10.3 PRUEBAS Y DEMO

Para hacer una demo real de la aplicación, sería necesario construir toda la arquitectura definida a la largo de este documento con todas las jerarquías y agentes involucrados corriendo y enviando mensajes. Sería necesario arrancar las vías, las estaciones y todos los dispositivos asociados a las mismas.

Es evidente que esta arquitectura no se puede montar para realizar la presentación del Proyecto Fin de Carrera. Lo que sí se puede hacer es realizar simuladores que emulen el comportamiento del sistema de la forma más ajustada posible a lo que sería su funcionamiento en producción, y eso es lo que se ha buscado.

Para ello, se va a implementar en la máquina local donde vaya a realizar la presentación y, por tanto, donde van a estar todos los componentes necesarios para el correcto funcionamiento de la aplicación, un simulador que va a generar los mensajes XML tal y como si estos provinieran de las vías.

Además, se instalará a nivel local un agente "Servidor de alarmas" que recogerá dichos mensajes de una cola configurada a tal efecto y que se los entregará al Servidor en Tiempo Real de modo que se puedan advertir en tiempo real los cambios de estado que se ordenen desde el simulador de mensajes en la aplicación web.

Para la presentación de aquellas funcionalidades de la Aplicación que no requieren una respuesta en tiempo real, se ejecutarán distintas pruebas que ilustren y verifiquen su correcto funcionamiento.

11 CONCLUSIONES Y TRABAJOS FUTUROS

Como se ha visto a lo largo de este documento, la aplicación de Supervisión del sistema de peaje es una herramienta que permite al usuario la visualización de información bajo demanda de forma completamente transparente a las particularidades del sistema de peaje en sí.

La arquitectura que subyace a la aplicación es perfectamente exportable a cualquier peaje, mientras que la aplicación se limita a recoger la información que le llega y a mostrarla en función de las características de configuración introducidas en sus Bases de Datos.

Este planteamiento da como resultado un sistema enormemente abierto y con grandes posibilidades de crecimiento. No hay más que pensar, por ejemplo, que si queremos ampliar la monitorización de las vías a más dispositivos, no tendremos más que agregar a las bases de datos dichos dispositivos, con sus correspondientes códigos de alarma y, por supuesto montar nuevos procesos capaces de monitorizar dichos dispositivos y generar los mensajes correspondientes ateniéndose al esquema diseñado.

Quiere decirse, pues, que el trabajo de ampliación a la posibilidad de monitorizar nuevas alarmas en nuevos dispositivos, requiere un trabajo externo que no exige un cambio en la aplicación. Como se ha dicho en varias ocasiones a lo largo de este documento, mientras la aplicación web tenga el mensaje correspondiente en el interfaz con la arquitectura de comunicaciones del peaje y se haya configurado convenientemente en la BBDD, la información se mostrará al usuario.

Además, la estructura del mensaje XML se ha definido para que pueda adaptarse a las necesidades de cualquier tipo de peaje e incluso anticipándose a posibles ampliaciones en el futuro.

Un trabajo ajeno a esta aplicación pero ligado con ella en el punto de generación de alarmas es lo que podría ser el desarrollo de un software de mantenimiento del sistema de peaje. Esto resultaría de enorme utilidad para la concesionaria explotadora de la autopista.

Un sistema de mantenimiento abarca muchos aspectos referentes al mantenimiento de la autopista.

La primera y más importante función del sistema de mantenimiento sería generar órdenes de trabajo que el personal de la autopista recibiría de inmediato en sus PDA cada vez que se genera una alarma en el sistema y la atendieran con un tiempo de respuesta lo más corto posible.

Se podría establecer que las alarmas de alta prioridad generasen órdenes de trabajo de tipo automático que no requieran la acción directa de un usuario, de manera que se tramiten la más rápido posible. Otras alarmas de prioridad baja deberían ser atendidas directamente por el personal a cargo que se encargaría de generar manualmente la orden de trabajo y asignársela al personal de mantenimiento.

Otras labores del sistema de mantenimiento son establecer políticas de mantenimiento preventivo y predictivo en base al análisis de los datos de las alarmas insertados en los registros de las bases de datos del sistema de Supervisión, partes de inventario, etc…

Como se ve, las posibilidades de la arquitectura planteada son enormes y el planteamiento que se le ha dado es lo menos dependiente posible de las particularidades de un peaje concreto. El objetivo es, obviamente, exportarlo a una gran cantidad de proyectos con el menor impacto posible y con las mínimas necesidades de nuevo desarrollo.

12 REFERENCIAS

- Direct Web Remoting http://directwebremoting.org
- Hibernate https://www.hibernate.org/
- Spring http://www.springsource.org/
- Struts http://struts.apache.org/
- Struts Tiles http://struts.apache.org/1.x/struts-tiles/
- Display Tag Library http://displaytag.sourceforge.net
- Yahoo User Interface http://developer.yahoo.com/yui/
- JAXB http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
- Apache Log4j http://logging.apache.org/log4j