Universitat Oberta de Catalunya

Segundo ciclo de Ingeniería Informática

Proyecto Fin de Carrera

"Gestión en el territorio de una empresa de seguridad mediante estándares SIG e integración de datos con Google Maps"

Memoria

Director: *Oscar Fonts Bartolomé*

Alumno: *Juan Luis Olmo Ortiz*

Curso 2006 – 2007, Semestre de primavera

Resumen

Con la evolución de las nuevas tecnologías, los Sistemas de Información Geográfica (SIG) se basan cada vez menos en plataformas propietarias (únicamente al alcance de grandes compañías). El objetivo que persigue el Open Geospatial Consortium (OGC) es que todo el mundo pueda beneficiarse de la información geográfica, haciendo que los servicios estén disponibles en cualquier red, aplicación o sistema.

La verdadera expansión de estos sistemas y su descubrimiento por el público general ha venido motivada por la posibilidad de que cualquier usuario, simplemente con tener conexión a Internet, pueda acceder a aplicaciones como Google Maps y Google Earth.

Este proyecto, en primer lugar, se centra en el estudio de las bases de los SIG, explicando sus componentes y utilidades. Asimismo, se presentan los formatos de información gráfica con los que los SIG pueden trabajar. Estos son los formatos en los que se basan los servicios web Web Map Service (WMS) y Web Feature Service (WFS), que también se explican en este proyecto, y que define el OGC para su implantación en la web. Este primer bloque del proyecto concluye con el estudio de la tecnología Google Maps.

El segundo bloque corresponde con el diseño y la implementación de una aplicación web que permite visualizar la situación geográfica de:

- Las cámaras de vigilancia de una empresa de seguridad, mostrando las imágenes que obtienen estas cámaras en tiempo real
- Los agentes de seguridad de la misma empresa de seguridad, mostrando también su ruta

Para este propósito, se desarrolla un mashup de Google Maps, esto es, se combina la información de la base de datos de la empresa con la información y servicios de Google Maps, mostrando finalmente la información en el visor de mapas.

Abstract

With the evolution of new technologies, the Geographical Information Systems are less based on proprietary platforms (only affordable by big companies). The Open Geospatial Consortium's (OGC) aim is that everybody can benefit from geographical information, making services available for any application or system across the network.

The increasing interest for these systems and its use by general public has been motivated by the possibility that any user, simply having an Internet connection, has access to applications as Google Maps and Google Earth.

In this project, on the one hand, we will focus on the study of the GIS bases, explaining its components and applications. We also present the main ways Geographical Information can be encoded. These are the formats on which two of the most widely used OGC standards Web Map Service (WMS) and Web Feature Service (WFS) are based, which are alse explained in this project, and the OGC defines for its implantation in the web. This first block of the project ends with the study of Google Maps technology.

The second block corresponds with the design and implementation of a web application that is able to show the geographical situation of:

- The video cameras of a security company, showing the images recorded in real time
- The company's security agents location and their route

For this purpose, a Google Map's mashup has been developed, this is, we combine the information of the corporate data base with the information and services of Google Maps, showing finally the information in the maps viewer.

Índice de contenidos

Índice de figuras

1 Introducción

1.1 Introducción y objetivos

Tradicionalmente los Sistemas de Información Geográfica (SIG, o GIS en inglés) se han caracterizado por ser aplicaciones complejas cuyo manejo y desarrollo requiere tener conocimientos específicos. Las aplicaciones SIG, al alcance únicamente de grandes empresas, se han basado en plataformas propietarias muy potentes pero cuya comunicación con otras aplicaciones se hace difícil. Este escenario tradicional ya no es el único posible.

La aparición de Google Maps y Google Earth ha abierto muchas posibilidades. Con estas aplicaciones cualquier usuario con una conexión a Internet puede moverse por el territorio y ver la imagen satélite de la superficie terrestre, las carreteras y el callejero de una buena cantidad de países, e incluso buscar elementos sobre el terreno de manera sencilla. Pero su potencial radica en que para los usuarios es posible integrar sus propios datos georreferenciados y dotar a las aplicaciones de nuevas funcionalidades.

Por otra parte, los SIG, como cualquier otro software corporativo, necesitan integrarse con otras aplicaciones de la empresa y comunicarse con distintas plataformas SIG. La industria se ha unido en el *Open Geospatial Consortium* (OGC) para consensuar una serie de estándares que permitan la interoperabilidad entre aplicaciones. A medida que se implantan estos estándares, surgen nuevas iniciativas, muchas de ellas basadas en código abierto, que encuentran su lugar en el mercado sin depender de formatos o protocolos de terceros.

Estos hechos abren nuevos escenarios donde es posible el desarrollo de aplicaciones flexibles y de bajo coste, que hacen uso del entorno web para integrar diversos orígenes de datos. Su límite está en que no están pensadas para manejar grandes volúmenes de datos ni efectúan análisis espaciales complejos. De hecho, como ya se planteará en este proyecto, cabría preguntarse si dichas aplicaciones pueden considerarse en realidad SIG.

El presente proyecto tiene como objetivo estudiar la implantación y potencial de estas nuevas herramientas que, sin tener las características de un SIG tradicional, son capaces de acercar las aplicaciones geográficas al gran público.

El proyecto se divide en dos bloques bien diferenciados:

Bloque 1: Estudio de herramientas y estándares existentes y de su implantación en la web. Este

bloque comprende los capítulos 1 a 5. Básicamente, se estudiarán:

a) **Servicios *Web Map Service* y *Web Feature Service* definidos por el OGC**, que permiten la recuperación de información cartográfica en formato ráster y vectorial.
b) **La API de Google Maps**, que permite la inclusión de un control de Google Maps en una aplicación web mediante el uso de JavaScript.

Bloque 2: Desarrollo de una aplicación web que permita visualizar la distribución en el territorio de los elementos de una empresa de seguridad. Integrará dos tipos de elementos a un visor de Google Maps:

a) **Videocámaras**: Deberá mostrarse la posición de cada una de ellas, y la imagen capturada en tiempo real de un subconjunto de ellas.
b) **Elementos móviles**: Deberá mostrarse y actualizarse en tiempo real la posición de cada uno de ellos, y su información asociada al seleccionarlo.

Los datos sobre estos elementos se extraerán de una base de datos, y se les deberá dar el formato adecuado para integrarlos en el control de Google Maps. Este segundo bloque se corresponde con los capítulos 5 y 6 de la memoria.

Este trabajo se centra en la consecución de 4 objetivos bien diferenciados:

- Conocer qué es un SIG
- Saber utilizar los estándares más extendidos del OGC: construir peticiones e interpretar posibles respuestas
- Descubrir los servicios y aplicaciones web existentes basadas en estándares abiertos (OGC) y de facto (Google). Valorar y comparar su implantación actual y potencial futuro.
- Desarrollar aplicaciones web complejas con Javascript, mediante la integración de servicios existentes.

1.2 Tareas, planificación, hitos

1.2.1 Tareas y actividades

A continuación describimos las tareas y actividades que habrá que realizar:

1 *Definición del proyecto*

1.1 Obtener la documentación inicial. Descarga del enunciado del proyecto y de la documentación de partida proporcionada por el consultor, así como otra información

consultada en la biblioteca (otros proyectos, etc).

1.2 Leer documentación inicial. Leer el enunciado y la información descargada.

2 *PEC1 – Plan de Trabajo*

2.1 Obtener la documentación necesaria. Descarga de la documentación de partida para la realización del Plan de Trabajo facilitada por el consultor.

2.2 Organizar documentación. Esquematizar y estructurar la información consultada, señalando las ideas clave.

2.3 Buscar bibliografía. Localización de bibliografía e información útil para la realización del proyecto a medio plazo, como sitios web, artículos, libros...

2.4 Definición del índice del Plan de Trabajo. Establecer los apartados que tendrá el Plan de Proyecto.

2.5 Redactar el Plan de Trabajo. Redactar la presentación, objetivos, estructura, actividades y tareas ...

2.6 Planificación. Establecer los hitos del proyecto y los contenidos de la PEC2 y la PEC3, así como el calendario de trabajo. Realizar el diagrama de Gantt teniendo en cuenta la duración estimada de las tareas y los posibles riesgos.

2.7 Terminar Plan de Trabajo. Finalizar el documento de Plan de Trabajo y entregarlo para su evaluación inicial.

2.8 Corregir Plan de Trabajo. Teniendo en cuenta las correcciones del consultor, completar el Plan de Trabajo o modificarlo en caso necesario.

2.9 Entregar Plan de Trabajo. Entregar el documento definitivo para su evaluación final (entrega PEC1).

3 *Estudiar los SIG*

3.1 Recopilar documentación sobre los SIG.

3.2 Estudiar los SIG. Conocer qué es un SIG, sus funciones y componentes, cómo se representa la información en los SIG, y qué aplicaciones tienen.

3.3 Redactar documentación.

4 *Estudiar los servicios web Web Map Service y Web Feature Service definidos por el OGC*

4.1 Recopilar información sobre el OGC y los servicios web a estudiar.

4.2 Estudiar documentación.

4.3 Redactar documentación.

5 *Estudiar Google Maps*

5.1 Recopilar información sobre Google Maps y las tecnologías en que se basa, buscar información sobre la API de Google Maps y buscar aplicaciones web que utilicen esta API.

5.2 Estudiar documentación.

5.3 Redactar documentación.

6 *Diseño de la aplicación web*

6.1 Definir claramente los requerimientos de la aplicación web.

6.2 Diseñar el modelo de datos de negocio.

6.3 Realizar el diseño arquitectónico.

6.4 Realizar el diseño de la interfaz.

6.5 Redactar documentación.

7 *Implementar la aplicación web*

7.1 Establecer el entorno de desarrollo necesario e instalar todo el software para implementar la aplicación.

7.2 Integrar al visor de Google Maps las videocámaras de nuestro modelo de datos.

7.3 Integrar al visor de Google Maps los elementos móviles.

7.4 Fase de pruebas. Realizar pruebas funcionales y unitarias.

7.5 Redactar documentación.

8 *Redactar memoria*

8.1 Redactar memoria a partir de toda la documentación ya redactada en las anteriores tareas.

8.2 Redactar conclusiones.

8.3 Redactar índice final.

8.4 Efectuar repaso de la memoria, poniendo especial atención en la ortografía.

9 *Confeccionar presentación*

9.1 Seleccionar contenidos más representativos de la memoria para incluirlos en la presentación.

9.2 Redactar presentación.

***10** Debate virtual*

1.2.2 Calendario de trabajo

En este apartado trataremos de establecer un calendario de trabajo semanal con las horas que se dedicarán a la realización del proyecto teniendo en cuenta otros factores como el horario laboral o los posibles días festivos.

La previsión de trabajo semanal queda como sigue:

- De lunes a viernes, 3 horas diarias.
- Sábados, 4 horas.
- Domingos, dedicados a recuperar planificación en caso de posibles retrasos (posibles reuniones de trabajo que impidan cumplir la planificación, etc). En caso de que la planificación se vaya cumpliendo, se tomarán de descanso.
- Días festivos, la dedicación al proyecto dependerá de las actividades familiares.

Esto supone una dedicación semanal de unas 19 horas, que podrían ser más en caso de que fuera necesario ocupar el domingo para recuperar la planificación.

A continuación se indicarán las horas totales de dedicación previstas por semana durante todo el semestre hasta la finalización del proyecto, indicando los días festivos:

Semana	*Horas*	*Días festivos o vacaciones*
1 (del 28 de febrero al 4 de marzo)	5	
1 (del 5 al 11 de marzo)	19	
2 (del 12 al 18 de marzo)	19	
3 (del 19 al 25 de marzo)	19	(22: aniversario)
4 (del 26 de marzo al 1 de abril)	19	
5 (del 2 al 8 de abril)	23	(5 y 6: jueves y viernes santo)
6 (del 9 al 15 de abril)	19	
7 (del 16 al 22 de abril)	22	
8 (del 23 al 29 de abril)	22	
9 (del 30 de abril al 6 de mayo)	22	(1: día del trabajo)
10 (del 7 al 13 de mayo)	22	

11 (del 14 al 20 de mayo)	22	
12 (del 21 al 27 de mayo)	13	(25 a 27: feria)
13 (del 28 de mayo al 3 de junio)	19	
14 (del 4 al 10 de junio)	19	
15 (del 11 al 17 de junio)	5	
15 (del 18 al 22 de junio)	5	

Como se puede observar, la dedicación en horas aproximada que requerirá el proyecto será de 280 horas.

Notas:
- En amarillo se indican las semanas que ya han pasado
- En la semana 3 se recuperarán las 4 horas del jueves 22 el domingo
- En la semana 5 se dedicarán 2 horas más al proyecto el jueves y el viernes
- En las semanas 7 a 11 se dedicarán 3 horas extra (seguras) todos los domingos, para la realización de la PEC3.
- En naranja se indican las semanas del debate virtual

1.2.3 Hitos

Se establecen los hitos del proyecto, que se corresponderán principalmente con las entregas parciales y final del proyecto. También se indica a la derecha la distribución de horas en base a la duración de cada hito (entregable).

Hito	*Fecha*	*Horas*
Inicio proyecto	28 de febrero	
Entrega Plan de Proyecto – PEC 1	12 de marzo	27
Entrega PEC2	16 de abril	99
Entrega PEC3	21 de mayo	116
Entrega final PFC, memoria y presentación	11 de junio	38
Fin proyecto (coincide con el final del debate virtual)	22 de junio	

1.2.4 Planificación

En este punto mostraremos la planificación del proyecto, indicando las tareas, subtareas, coste en horas y fechas de inicio y fin. También se indican las entregas de las diferentes PECs en la planificación.

Tarea	Nombre	Actividades	Horas	Inicio	Fin
1	Definición del proyecto		5	28 feb	4 mar
		Obtener la documentación inicial			
		Leer documentación inicial			
2	PEC1 – Plan de Trabajo		22	5 mar	12 mar
		Obtener la documentación necesaria	2	5 mar	
		Organizar documentación	4	5 mar	6 mar
		Buscar bibliografía	2	6 mar	
		Definición del índice del Plan de Trabajo	3	6 mar	7 mar
		Redactar el Plan de Trabajo	3	7 mar	8 mar
		Planificación	5	8 mar	9 mar
		Terminar Plan de Trabajo	1	9 mar	9 mar
		Corregir Plan de Trabajo	2	10 mar	10 mar
		Entregar Plan de Trabajo		**12 mar**	**12 mar**
3	Estudiar los SIG		19		
		Recopilar información	2	13 mar	13 mar
		Estudiar los SIG	7	13 mar	15 mar
		Redactar documentación	10	16 mar	19 mar
4	Estudiar WMS y WFS		32		
		Recopilar información	3	20 mar	20 mar
		Estudiar los servicios web	10	21 mar	24 mar
		Redactar documentación	19	26 mar	31 mar
5	Estudiar Google Maps		48		

		Recopilar información	6	2 abr	3 abr
		Estudiar Google Maps	19	4 abr	9 abr
		Redactar documentación	21	9 abr	16 abr
		Entregar PEC2		**16 abr**	**16 abr**
6	Diseño de la aplicación web		47		
		Definir requerimientos	6	17 abr	18 abr
		Diseñar modelo de datos	13	19 abr	22 abr
		Diseño arquitectónico	6	23 abr	24 abr
		Diseño de la interfaz	5	25 abr	26 abr
		Redactar documentación	17	26 abr	2 may
7	Implementar la aplicación web		73		
		Entorno desarrollo	6	3 may	4 may
		Integrar videocámaras a Google Maps	29	5 may	12 may
		Integrar elementos móviles a Google Maps	22	13 may	19 may
		Pruebas	4	20 may	20 may
		Redactar documentación	12	21 may	24 may
		Entregar PEC3		**24 may**	**24 may**
8	Redactar memoria		27		
		Redactar memoria a partir de documentación ya redactada	15	25 may	1 jun
		Redactar conclusiones	3	2 jun	2 jun
		Redactar índice final	3	3 jun	3 jun
		Repaso memoria	6	4 jun	5 jun
9	Confeccionar presentación		16		
		Seleccionar contenidos	6	6 jun	7 jun

		Redactar presentación	10	8 jun	11 jun
		Entrega memoria y presentación		**11 jun**	**11 jun**
10	Debate virtual		7	12 jun	22 jun

1.2.4.1 Diagrama de Gantt para la PEC1

A continuación mostramos el diagrama de Gantt correspondiente a la PEC1

	Nombre de tarea	Duración	Comienzo
1	Inicio proyecto	0 días	mié 28/02/07
2	**– Definición del proyecto**	**4 días?**	**mié 28/02/07**
3	Obtener la documentación inicial	1 día?	mié 28/02/07
4	Leer documentación inicial	3 días?	jue 01/03/07
5	**– PEC1 - Plan de Trabajo**	**7 días?**	**lun 05/03/07**
6	Obtener la documentación necesar	1 día?	lun 05/03/07
7	Organizar documentación	2 días?	lun 05/03/07
8	Buscar bibliografía	1 día?	mar 06/03/07
9	Definición del índice del Plan de Tra	2 días?	mar 06/03/07
10	Redactar el Plan de Trabajo	2 días?	mié 07/03/07
11	Planificación	2 días?	jue 08/03/07
12	Terminar Plan de Trabajo	1 día?	vie 09/03/07
13	Corregir Plan de Trabajo	1 día?	sáb 10/03/07
14	Entregar Plan de Trabajo	1 día?	lun 12/03/07

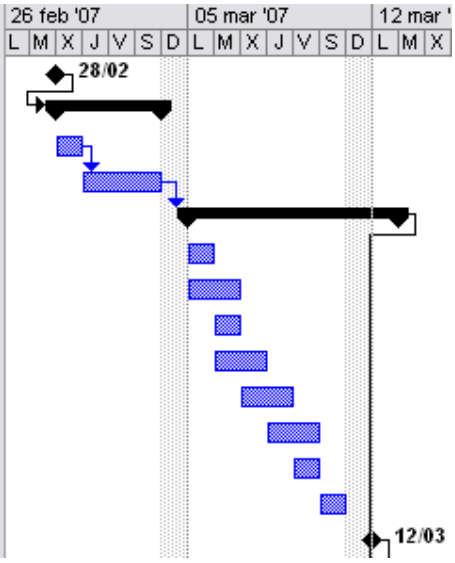

Figura 1.1. Diagrama de Gantt PEC1

1.2.4.2 Diagrama de Gantt para la PEC2

En la figura adjunta podemos observar el diagrama de Gantt correspondiente a la PEC2:

15	– PEC2	29 días?	mar 13/03/07
16	– Estudiar los SIG	6 días?	mar 13/03/07
17	Recopilar información	1 día?	mar 13/03/07
18	Estudiar los SIG	3 días?	mar 13/03/07
19	Redactar documentación	2 días?	sáb 17/03/07
20	– Estudiar WMS y WFS	10 días?	mar 20/03/07
21	Recopilar información	1 día?	mar 20/03/07
22	Estudiar los servicios web	3 días?	mié 21/03/07
23	Redactar documentación	6 días?	lun 26/03/07
24	– Estudiar Google Maps	13 días?	lun 02/04/07
25	Recopilar información	2 días?	lun 02/04/07
26	Estudiar Google Maps	4 días?	jue 05/04/07
27	Redactar documentación	6 días?	mar 10/04/07
28	Entregar PEC2	1 día?	lun 16/04/07

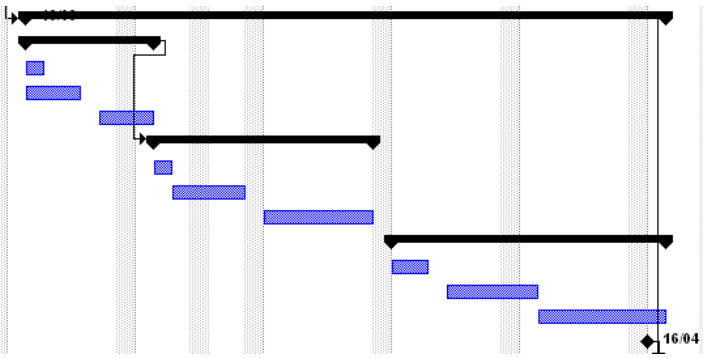

Figura 1.2. Diagrama de Gantt PEC2

1.2.4.3 Diagrama de Gantt para la PEC3

La siguiente figura se corresponde con el diagrama de Gantt de la PEC3.

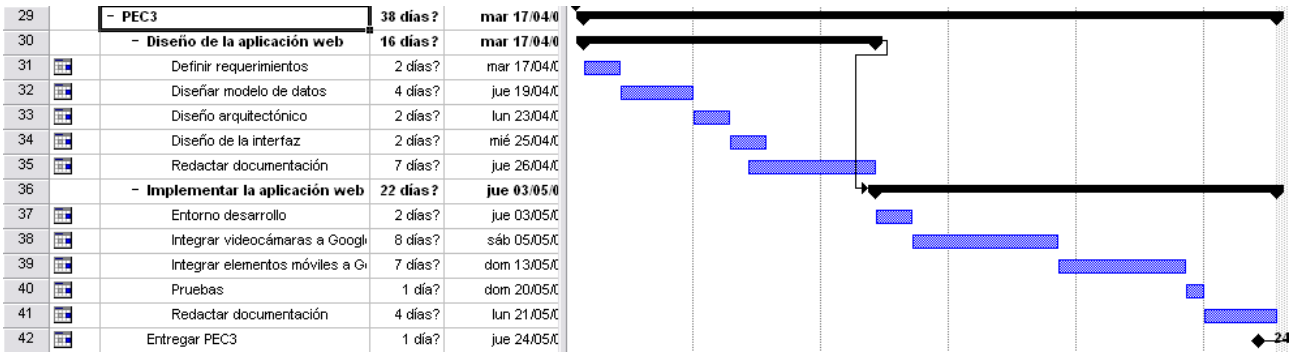

Figura 1.3. Diagrama de Gantt PEC3

1.2.4.4 Diagrama de Gantt para la entrega final del PFC

Por último se muestra el diagrama de Gantt correspondiente a la memoria y presentación, es decir, la entrega final del PFC, y el debate virtual.

Figura 1.4. Diagrama de Gantt entrega final

1.3 Análisis de riesgos y plan de contingencia

En este apartado nos centraremos en prever los posibles riesgos que pueden surgir durante la realización de este proyecto y que pueden afectar a su planificación. Se describen las medidas a tomar para minimizar los efectos de dichos riesgos y poder cumplir la planificación.

- Avería de la estación de trabajo. Es posible que el equipo donde desarrollamos el proyecto sufra una avería. Como medida preventiva, iremos realizando copias de seguridad periódicas de toda la información del proyecto. No obstante, en caso de avería, tendríamos que:
  - Conseguir otro equipo en el que trabajar o arreglar la avería.
  - Instalar todo el software necesario y configurarlo: entorno de desarrollo, servidor de bases de datos...

- Efectuar horas extras en el trabajo. En caso de que la empresa tenga problemas y nos solicite efectuar horas extras, trataremos de terminar el trabajo lo más rápidamente posible. En caso necesario se usará el tiempo necesario del domingo, que se tomó como día libre.

1.4 Productos obtenidos

El proyecto de Fin de Carrera (PFC) consiste en la entrega de los siguientes elementos:

- **Memoria**: documento que recoge todo el trabajo realizado, de un máximo de 90 páginas
- **Presentación**: resumen autoexplicativo del trabajo realizado, de un máximo de 20 diapositivas
- **Trabajo práctico o producto software**: incluirá el código fuente e instrucciones para la instalación y puesta en producción del mismo

1.5 Descripción de capítulos

La presente memoria se estructura en 6 capítulos:

- **Capítulo 1: "Introducción"**. Se presenta el proyecto, realizando una introducción al mismo y a los objetivos que se pretenden alcanzar. Se indica la planificación a seguir con el tiempo de dedicación a cada tarea, el estudio de riesgos y plan de contingencia, así como los productos finales.
- **Capítulo 2: "Introducción a los SIG"**. En este capítulo se introduce al lector en el concepto de SIG, explicando sus principales funciones, componentes y aplicaciones. También se explica cómo se representa la información en un SIG: en formato ráster o en formato vectorial. Por último, se compara a los SIG con otros sistemas con los que comparte muchos rasgos en común.
- **Capítulo 3: "Servicios web definidos por el OGC"**. Se introduce a la labor del OGC en el desarrollo de estándares y especificaciones de interoperabilidad para datos espaciales. El capítulo se centra en el estudio de dos estándares que se implantan en la web, los servicios WMS y WFS, cuyo objetivo es permitir la obtención de información cartográfica en formato

ráster y vectorial, respectivamente.

- **Capítulo 4: "Google Maps"**. Este capítulo se inicia con una introducción a la tecnología Google Maps, su historia y AJAX; también se realiza una breve reflexión sobre si es correcto considerar a Google Maps como un SIG. A continuación, el capítulo se centra en el estudio de la API de Google Maps y sus objetos básicos mediante ejemplos sencillos y representativos. Los objetos de que se compone la API se han clasificado en 5 categorías: núcleo, controles de mapa, datos de usuario, eventos y AJAX. Finalmente, se muestran aplicaciones/mashups de Google Maps, y se indica cómo se puede modificar la API para mostrar capas WMS sobre el visor.
- **Capítulo 5: "Diseño de la aplicación web"**. En este capítulo se efectúa el análisis de la información que ha de mantener la aplicación web, mediante modelado de datos UML. A partir del diagrama de clases obtenido, se realiza la traducción al modelo relacional y se genera el script de creación de la estructura de la base de datos de la aplicación. En los apartados que siguen, se muestra la arquitectura de la aplicación y se describe la interfaz de usuario, realizando un análisis de la ergonomía y la usabilidad de la misma.
- **Capítulo 6: "Implementación de la aplicación web"**. En este capítulo se describe el entorno de desarrollo utilizado en la implementación de la aplicación, se describe características comunes del código y se explica por separado cómo se ha implementado la integración de cámaras al visor de Google Maps y cómo se han integrado los elementos móviles al mismo, usando la tecnología AJAX. Se adjuntan figuras para explicar el funcionamiento del paso de mensajes entre el navegador del cliente y el servidor web, pasando por el motor de AJAX.

Como se puede observar, los cuatro primeros capítulos se corresponden con el primer bloque del proyecto, referente al estudio de herramientas y estándares existentes y de su implantación en la web. Los capítulos 5 y 6, en cambio, son los correspondientes al segundo bloque del proyecto, es decir, el desarrollo de una aplicación web que permita visualizar la distribución en el territorio de los elementos de una empresa de seguridad, integrando videocámaras y elementos móviles a un visor de Google Maps.

2 Introducción a los Sistemas de Información Geográfica

2.1 ¿Qué es un SIG?

En primer lugar se explicará qué se entiende por Información Geográfica (IG): se denomina Información Geográfica a aquellos datos espaciales georreferenciados requeridos como parte de las operaciones científicas, administrativas o legales. Dichos datos espaciales suelen llevar una información alfanumérica asociada.

Un SIG o Sistema de Información Geográfica, por tanto, funciona como una base de datos con información geográfica (datos alfanuméricos) que se encuentra asociada por un identificador común a los objetos gráficos de un mapa digital. De esta forma, señalando un objeto se conocen sus atributos y, de forma inversa, preguntando por un registro de la base de datos se puede saber su localización en la cartografía.

Un SIG separa la información en diferentes capas temáticas y las almacena independientemente, permitiendo trabajar con ellas rápida y fácilmente, y posibilitando relacionar la información existente en las diferentes capas a través de la topología de los objetos, con el fin de generar otra nueva capa imposible de obtener de otro modo.

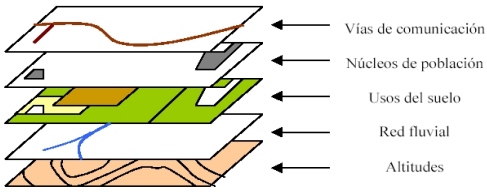

Figura 2.1. Capas temáticas de un SIG

En un SIG, la topología se refiere a las relaciones espaciales entre los diferentes elementos gráficos y su posición en el mapa. Estas relaciones, que para el ojo humano pueden resultar obvias, el software debe establecerlas mediante un lenguaje y unas reglas de geometría matemática. Es la capacidad de crear topología lo que diferencia a un SIG de otros sistemas de gestión de la información.

Un SIG es un sistema informatizado que utiliza información locacional, tal como domicilios, distritos electorales, números de lotes o coordenadas de longitud y latitud, para mapear información para un mejor análisis. Con un SIG, uno puede mapear clientes para estudiar demografía, estudiar cómo se dispersa una enfermedad buscando patrones, modelar el paso de la contaminación atmosférica, etc.
Una definición más técnica puede ser la dada por Aronnof (1989) *[02_01]*, que dice así: "*Un Sistema de Información Geográfica es un sistema basado en el ordenador (CBS) que proporciona los cuatro conjuntos de capacidades siguientes para el manejo de datos georreferenciados: entrada de datos, gestión de los datos, manipulación y análisis, y salida de información*". Esta definición está basada en

las funciones que son necesarias en un SIG, que trataremos en el apartado siguiente.

Los SIG trabajan con una metodología propia y poseen un núcleo técnico importante en el que se combinan conceptos de diferentes disciplinas (topología, estadística espacial, geometría computacional, ...).

2.2 Funciones de los SIG

Una posible clasificación de las funciones que tiene un Sistema de Información Geográfica puede ser la siguiente:

- Funciones para la entrada de información: son aquellas que permiten la conversión de la información geográfica del formato real analógico al formato digital sin pérdida de información esencial, incluyendo los procedimientos de eliminación de errores y la digitalización.
- Funciones para la salida de información: son aquellas funciones que permiten mostrar al usuario los datos introducidos y el análisis de los mismos. Los SIG pueden hacerlo de diversas formas (gráficos, mapas, tablas), y en diferentes soportes (papel, pantalla, etc.).
- Funciones de gestión de la información espacial: son las funciones relativas a la extracción de la base del subconconjunto de datos que interesa en un momento concreto.
- Funciones analíticas: son aquellas que aumentan el conocimiento derivado de los datos. A esto es a lo que se denomina análisis espacial.

En base a estas funciones básicas, los SIG pueden resolver cuestiones como *[02_02]*:

- Localización: preguntar por las características de un lugar concreto.
- Condición: el cumplimiento o no de unas condiciones impuestas al sistema.
- Tendencia: comparación entre situaciones temporales o espaciales distintas de alguna característica.
- Rutas: cálculo de rutas óptimas entre dos o más puntos.
- Pautas: detección de pautas espaciales.
- Modelos: generación de modelos a partir de fenómenos o situaciones simuladas.

2.3 Componentes de un SIG

Los componentes básicos de un SIG son *[02_03] [02_04]*:

- Hardware: los SIG se ejecutan en un amplio rango de tipos de computadoras (desde equipos centralizados hasta configuraciones individuales o de red). Componentes hardware serán las

máquinas donde corren el sistema gestor de bases de datos, las aplicaciones de la interfaz gráfica, los equipos que obtienen información geográfica, etc. .

- Software: los programas SIG proveen las herramientas y funcionalidades necesarias para almacenar, analizar y mostrar información geográfica. Los componentes principales del software SIG son:
  - Sistema gestor de bases de datos.
  - Interfaz gráfica de usuario (GUI) para interactuar de forma fácil e intuitiva.
  - Herramientas para captura y manejo de información geográfica.
  - Herramientas para soporte de consultas, análisis y visualización de datos geográficos.
- Información: es el componente más importante de un SIG, puesto que son necesarios datos de soporte adecuados para que el SIG pueda resolver problemas y contestar a preguntas de la forma más adecuada posible. Los datos geográficos y alfanuméricos pueden obtenerse mediante recursos propios o por medio de proveedores de datos (externos u organismos oficiales).
- Personal: las tecnologías SIG son de valor limitado si no se cuenta con especialistas que manejen el sistema y exploten el máximo potencial del hardware y el software.
- Métodos: para que un SIG tenga una implementación exitosa debe basarse en un buen diseño y reglas de actividad definidas, que son los modelos y prácticas operativas exclusivas en cada organización.

2.4 Representación de la información en un SIG

Los proyectos SIG son únicos en el hecho de que dependen de tener datos existentes del lugar. Estos datos preexistentes, llamados datos de mapas base o base cartográfica (*basemap data*), generalmente los crea y los mantiene alguien más. Nuestro trabajo como desarrolladores SIG será pues encontrar estos datos e incorporarlos en el producto final.

Para entender el concepto de *basemap*, podemos pensar en un parte meteorológico en televisión. Supongamos que el presentador está de pie delante de un sistema tormentoso que da vueltas (la capa de datos) sobrepuesto sobre ciudades y caminos (la cartografía de base).

En términos de programación, las aplicaciones GIS son una serie de mapas sumamente cohesivos que se acoplan. La idea es acoplar estos mapas de maneras nuevas e interesantes para producir nueva información.

En cuanto a una capa de mapa, hay dos tipos fundamentales de datos a considerar: datos de ráster y datos vectoriales. Un SIG puede almacenar la información en formato ráster o vectorial:

- Los datos **ráster** son una matriz de puntos o malla regular que representan un muestreo de alguna característica del territorio. Un ejemplo de datos ráster puede ser una imagen satélite de la Tierra o tan sólo una fotografía aérea. Es decir, el modelo de SIG ráster o de retícula se centra más en las propiedades del espacio que en la precisión de localización. De forma más técnica, un ráster es un archivo que almacena sus datos en celdas discretas organizadas en filas y columnas (podríamos pensar en algo similar a una hoja de cálculo, sólo que en este caso las celdas individuales son píxeles de una fotografía). Cuanto mayores sean las dimensiones de las celdas (resolución) menor es la precisión o detalle en la representación del espacio geográfico.
- En el modelo de SIG **vectorial**, el interés se centra en las representaciones geométricas de los elementos y las relaciones topológicas de los mismos, que permiten al SIG aplicar funciones analíticas (análisis espacial) que no es posible realizar con los datos ráster. Para modelar digitalmente las entidades del mundo real se utilizan tres elementos espaciales: el punto, la línea y el polígono.

A continuación veremos en detalle ambos modelos y en qué contextos son más útiles.

2.4.1 El modelo vectorial

La cuestión de cuándo usar datos ráster o vectoriales no es una cuestión de cuál es cualitativamente mejor, sino de cuál es más apropiado para el contexto que estamos tratando (representar los datos o analizarlos).

Antes dijimos que el modelo ráster almacena valores en celdas discretas. Cada píxel de una fotografía almacena un valor específico. El modelo vectorial difiere en que sólo almacena vértices, esto es, almacena cada esquina en lugar de las líneas enteras. Esto hace que el formato sea mucho más compacto, pero sólo es apropiado para datos que no requieran valores discretos. La idea de usar uno u otro modelo, por tanto, sería la siguiente: el modelo vectorial es más apropiado para guardar los contornos de los objetos, mientras que el ráster es más adecuado para expresar el volumen o el contenido del objeto.

Hay tres tipos básicos de datos vectoriales:

- **Puntos** (OD): sirven para representar pozos, buzones, ...
- **Líneas** (1D): carreteras, ríos, rutas de transportes, elevación del terreno...
- **Polígonos** (2D): continentes, países, lagos, términos municipales, ...

A continuación presentamos dos imágenes para ejemplificar el modelo vectorial. En la primera vemos una captura del SIG de código libre gvSIG donde se está editando una capa vectorial de polígonos. En ella podemos ver que la base es una ortofotografía (ráster), y que sobre la misma se presentan líneas y polígonos (procedentes de una capa vector).

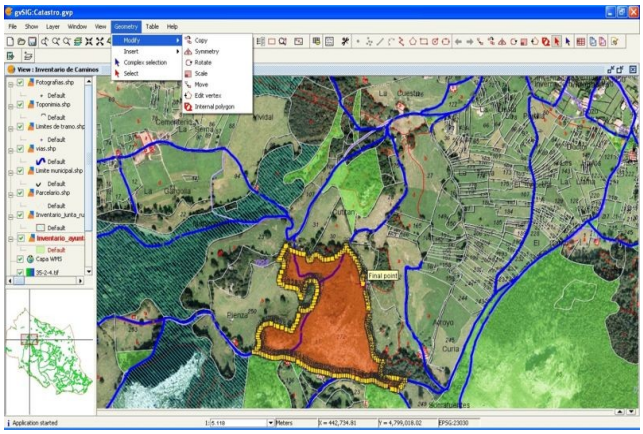

Figura 2.2. Captura gvSIG capa vectorial

La segunda imagen se corresponde con una captura del SIG Google Earth, para la cuál la Dirección General del Catastro ha desarrollado una utilidad para poder consultar y visualizar toda la cartografía catastral.

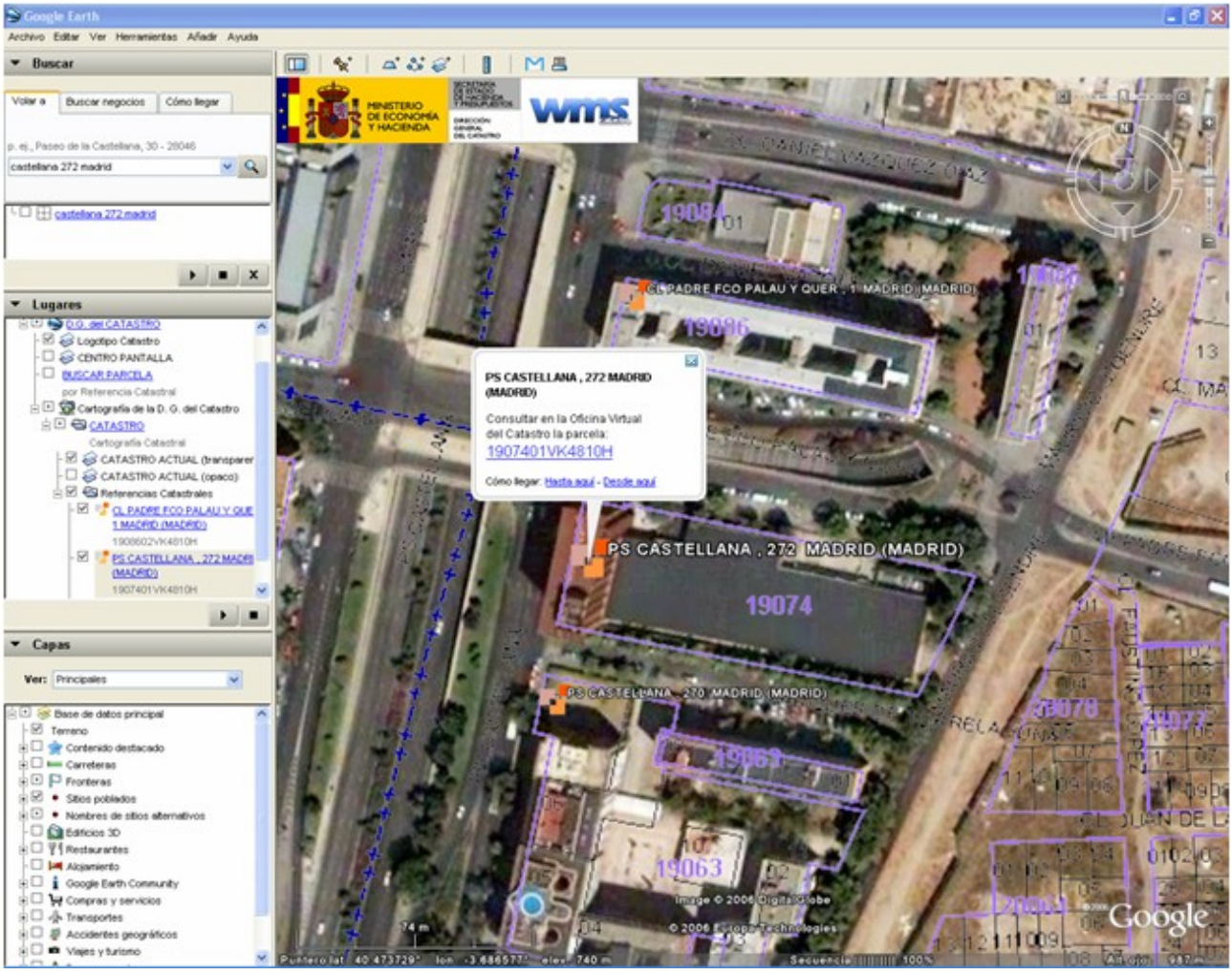

Figura 2.3. Captura Google Earth capa vectorial

2.4.2 El modelo ráster

Ya que en este proyecto se estudia la aplicación Google Maps, para comenzar el estudio del modelo ráster analizaremos en primer lugar el sitio web http://maps.google.es. Al acceder a esta dirección, se puede observar una vista de España en modelo vectorial; pero si se pulsa en el botón "Satélite", entonces se muestra una vista ráster de España de las mismas dimensiones que la inicial. En los copyrights de la parte inferior derecha del visor de Google Maps, se puede ver que en este nivel de zoom Google ha comprado la imagen a la NASA (tomada desde alguno de sus satélites).

En el modelo ráster la imagen se divide en una retícula o malla regular de pequeñas celdas o píxeles, como dijimos anteriormente, y se atribuye un valor numérico a cada celda como representación de su valor temático. Al ser la malla regular y por tanto el tamaño del pixel constante, y al conocer la posición en coordenadas del centro de cada una de las celdas, se puede decir que todos los pixels están georreferenciados.

Un SIG almacenará estos valores en una base de datos *[02_05]*, y, si deseamos tener una descripción precisa de los objetos geográficos contenidos en la misma, el tamaño del pixel ha de ser reducido (en función de la escala), lo que dotará a la malla de una resolución alta. Sin embargo, a mayor número de filas y columnas en la malla (más resolución), supondrá un mayor esfuerzo el proceso de captura de la información y un mayor costo computacional a la hora de procesar la misma.

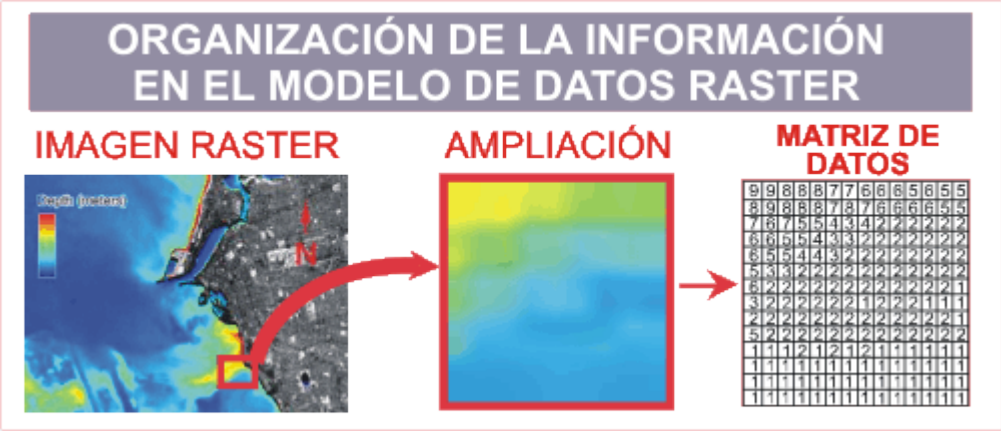

Figura 2.4. Modelo ráster

No obstante, el modelo de datos ráster es especialmente útil cuando tenemos que describir objetos geográficos con límites difusos, como pueden ser los niveles de contaminación de un acuífero subterráneo, donde los contornos no son absolutamente nítidos; en estos casos, el modelo ráster es más apropiado que el vectorial.

2.4.3 Comparación entre ambos métodos

En este apartado enumeramos las ventajas e inconvenientes de ambos métodos:

Ventajas del modelo vectorial:

- Define fronteras explícitamente (registra los límites entre entidades)
- Ocupa menor espacio de almacenamiento
- Proporciona una cartografía de calidad. Es más adecuado para generar salidas gráficas que se aproximan mucho a los mapas dibujados a mano
- Guarda una codificación eficiente de la topología y, por tanto, una implementación más eficiente de las operaciones que requieren información topológica (como el análisis de redes)

Inconvenientes del modelo vectorial:

- Presenta una estructura de datos más compleja
- Las operaciones de superposición de mapas son más difíciles de implementar
- La simulación y el análisis sobre el espacio son más complejos, al tratar con vectores
- El hardware sobre el que corre el SIG ha de ser más potente, al tratar con vectores

Ventajas del modelo ráster:

- Permite una superposición fácil y rápida, siempre que superpongamos dos rásters con el mismo tamaño de píxel. El análisis y la simulación en este caso son más sencillos, al estar los elementos a dibujar contenidos en píxeles.
- En principio, se puede pensar que el hardware no tiene que ser demasiado potente, ya que tan sólo tiene que poder tratar con imágenes. Sin embargo, se requiere trabajar con un gran volumen de información, sobre todo en el caso de que las imágenes ocupen mucho espacio y se estén superponiendo o realizando análisis, lo cual hace que el hardware necesariamente sea potente.
- Es adecuado para modelado espacial
- La cartografía es más económica

Inconvenientes del modelo ráster:

- Presenta un gran volumen de datos. Tiene necesidades de espacio considerables.
- Para que las imágenes no ocupen demasiado espacio, lo normal es que el tamaño de la celda no sea lo bastante pequeño, de forma que presenta menor resolución y mayor dificultad de estructuración en capas de información
- La forma de los elementos no queda clara, por tanto no se pueden realizar consultas implícitas utilizando los datos de la proximidad. De hecho, la salida de los gráficos resulta menos estética, ya que los límites entre zonas tienden a presentar la apariencia de bloques en comparación con las líneas suavizadas de los mapas dibujados a mano. Esto puede

solucionarse usando un elevado número de celdas más pequeñas, pero el problema es que pueden resultar ficheros inaceptablemente grandes

2.5 Aplicaciones de los SIG

Los SIG se aplican en gran cantidad de campos dado que el interés en el manejo de la información geográfica es aplicable en múltiples ámbitos. A continuación citamos algunas aplicaciones *[02_04]*:

- Producción y actualización de la cartografía básica
- Administración de servicios públicos (alcantarillado, energía, teléfono...)
- Atención de emergencias (incendios, terremotos, accidentes, ...)
- Estudios sociológicos y demográficos
- Gestión de recursos naturales
- Gestión del tráfico y diseño y mantenimiento de la red vial (radares, carreteras, peajes, rutas, ...)
- Evaluación de áreas de riesgos (prevención y atención de desastres)
- Localización geográfica (gestión de personas, logística, transporte, ...), integración con productos GPS
- Geomárketing (que permite analizar la situación de un negocio apoyándose en la variable espacial: situación de los clientes, puntos de venta, sucursales, competencia, etc.)

2.6 Comparación con otros sistemas

Existe una cierta dificultad *[02_06]* para fijar los límites de los SIG con respecto a otras herramientas informáticas, como el CAD (*Computer-Aided Design*, Diseño Asistido por Computador), la cartografía automática, los SGBDs (Sistemas de Gestión de Bases de Datos) y los sistemas de teledetección (tratamiento de imágenes de satélite). Todos ellos son anteriores en el tiempo a los SIG. Dado que los SIG han evolucionado a partir de esos sistemas, poseen muchos rasgos en común con ellos, pero también ciertos rasgos diferenciales.

2.6.1 SIG vs CAD

Los sistemas CAD, nacieron para diseñar y dibujar nuevos objetos. Son herramientas muy utilizadas por diseñadores, delineantes, arquitectos e ingenieros. El objetivo son las funcionalidades gráficas. Por ello pronto fueron utilizados para dibujar mapas, que se estructuraban en capas temáticas, mejorando el proceso de producción. Para Burrough(1986) *[02_07]* la mayor diferencia entre los

sistemas SIG y CAD estriba en el volumen y diversidad de datos mucho mayor que maneja el SIG y en los métodos de análisis que utiliza (es decir, la base de datos y el análisis espacial). Lo que distingue al SIG del CAD es la capacidad de aquel para integrar datos georreferenciados y para realizar ciertas operaciones de análisis, como la búsqueda espacial (que incluye el análisis de proximidad o *buffer*) y las superposiciones de mapas.

2.6.2 SIG vs Cartografía Automática

Los sistemas de cartografía automática ofrecen grandes ventajas a la hora de realizar cartografía de alta calidad. El punto de interés se sitúa en el dibujo de los mapas, pero no en el análisis. La principal diferencia con respecto a los SIG estriba en que los sistemas de cartografía automática no generan topología, los mapas son simplemente dibujos. Estos mapas digitales son asimilables a trasparencias que se pueden superponer. Se puede ver dónde se produce un cruce de carreteras general y otra secundaria, pero el sistema no tiene conocimiento de ello. La geometría está presente, pero la topología y la conectividad de la red están ausentes. Por otra parte, aunque estos sistemas puedan conectarse a las bases de datos, éstas no constituyen una parte esencial de ellos. El concepto de base de datos es básico y constituye para algunos la principal diferencia entre un SIG y un sistema de confección de mapas informatizados, el cual sólo puede producir mapas de calidad.

2.6.3 SIG vs Sistemas Gestores de Bases de Datos (SGBDs)

Los SGBD son sistemas desarrollados para almacenar y tratar información alfanumérica. Pueden tratar grandes volúmenes de información, pero apenas poseen funcionalidades gráficas. Evidentemente constituyen un componente esencial de los SIG, pero solo cuando están especialmente diseñados para trabajar con información espacial. En este caso hablaríamos de SGBDs espaciales, preparados para manejar datos espaciales, cuyo lenguaje de consulta es el SSQL (Spatial Structured Query Language), el cual introduce, mediante extensiones, los distintos conceptos del álgebra basada en los tipos de datos espaciales reales dentro del lenguaje SQL estándar. Ejemplos de estos SGBDs pueden ser PostGIS (ampliación de PostgreSQL que soporta objetos geográficos), Oracle Spatial o ArcSDE.

2.6.4 SIG vs Teledetección

Los sistemas para el tratamiento de imágenes satélite constituyen un campo cada vez más próximo al de los SIG, de manera que hoy día la teledetección se considera como una fuente para alimentar a los

Sistemas de Información Geográfica. Estos sistemas están diseñados para tratar la información obtenida a partir de sensores remotos, que son capaces de captar la radiación que emite la superficie terrestre. Ponen un especial énfasis en las operaciones de clasificación de esos datos, pero sus capacidades de análisis suelen ser reducidas. Algunos de estos sistemas pueden conectarse a un SIG para la realización de posteriores análisis o incluso ambos sistemas pueden estar plenamente integrados en un mismo producto. Es importante la complementariedad de ambas tecnologías aunque se desarrollaran de forma separada desde un principio.

3 Servicios web definidos por el OGC

3.1 Introducción al OGC

El Open Geospatial Consortium (OGC) *[03_01]* es una organización sin ánimo de lucro, internacional y con miembros de diferentes instituciones que encabeza el desarrollo de estándares para sistemas que procesan datos georreferenciados (espaciales).

El objetivo es crear especificaciones para la interoperabilidad a nivel de interfaz entre componentes, para que puedan intercambiar información geográfica. De esta forma, todo el mundo puede beneficiarse de la información geográfica y los servicios estarían disponibles a través de cualquier red, aplicación o sistema.

Actualmente el OGC ofrece especificaciones formales de interoperabilidad de datos espaciales para diferentes ámbitos de implementación (CORBA, COM, SQL, WEB):

- Acceso de las entidades simples (Simple Features – SQL, CORBA, OLE)
- Servicios de Catálogo
- Coberturas
- Servicios de Transformación de Coordenadas
- Servicios de Mapas por el Web (*Web Map Service,* WMS)
- Lenguaje de Marcado Geográfico (*Geographic Markup Language,* GML)
- Servicio Web de Objetos Geográficos (*Web Feature Service,* WFS)
- Descriptor de Estilos de Capas (*Styled Layer Descriptor*, SLD), un lenguaje para producir mapas georreferenciados con los estilos definidos por el usuario

La mayor parte de las empresas líderes del sector de Sistemas de Información Geográfica son miembros de esta organización y contribuyen a la producción de especificaciones y estándares, a la vez que los aplican a sus productos comerciales (Oracle, MapInfo, Intergraph, ESRI, Sun, Bentley, SmallworldWide, ...). También forman parte activa de esta organización gobiernos y miembros de diferentes universidades.

En los dos apartados siguientes nos centraremos en estudiar dos de los estándares definidos por el OGC que se implantan en la web, los servicios Web Map Service y Web Feature Service, cuyo objetivo es permitir la obtención de información cartográfica en formato ráster y vectorial, respectivamente, desde servidores web específicos compatibles, y combinar dicha información de modo que pueda ser mostrada y manipulada por un navegador web estándar.

3.2 Web Map Service

3.2.1 Descripción

El Open Geospatial Consortium Web Map Service (OGC-WMS, ahora ISO19128) *[03_02]* es una especificación tecnológica que permite obtener mapas desde un servidor específico compatible con WMS. Hablamos de "mapa" en este contexto como de representaciones visuales de los datos geoespaciales (no de los propios datos geoespaciales). Estos mapas son representaciones estáticas (imágenes) de una base geoespacial específica (están geoespacialmente referenciadas), es decir, estamos hablando de vistas ráster.

El estándar define tres operaciones *[03_03]*:

- Devolver metadatos del nivel del servicio (petición `GetCapabilities`)
- Devolver un mapa cuyos parámetros geográficos y dimensionales han sido bien definidos (petición `GetMap`)
- Devolver información de características particulares mostradas en el mapa (petición `GetFeatureInfo`)

Hay dos tipos de WMS:

- Uno es el **WMS básico**, que proporciona capas de mapas en estilos predefinidos, y donde el cliente selecciona el estilo de una lista de estilos predefinidos.
- El otro tipo de WMS es el llamado **SLD**, *Styled Layer Descriptor*, que necesita que internamente la información sea vectorial para así poder distinguir los distintos elementos. Por ejemplo, puede trabajar conjuntamente con el WFS: en este caso, el WMS funciona como un instrumento de interpretación, mientras que el WFS proporciona la fuente de datos. Así, el cliente tiene acceso a una biblioteca de estilos y símbolos, y puede "instruir" al WMS en los estilos que va a utilizar.

Ambos tipos de WMS pueden servir datos geoespaciales en dos formatos, imágenes o bien elementos gráficos. Los formatos de imagen incluyen formatos comunes como GIF (Graphics Interchange Format), PNG (Portable Network Graphics), JPEG (Joint Photographics Expert Group). Los formatos con elementos gráficos pueden ser SVG (Scalable Vector Graphics) o WebCGM (Web Computer Graphics Metafile). El OGC no fuerza a usar ningún formato específico, no obstante sí que recomienda usar un formato que proporciona transparencia, en el caso de que se pretendan superponer mapas o se quiera obtener un mapa con múltiples capas.

La mayoría de los navegadores de mapas cliente representan estos mapas en ventanas estáticas donde, en algunos casos, es posible superponer capas diferentes que provengan de servidores distintos

(interoperabilidad).

La especificación establece un lenguaje de comunicación entre un cliente y un servidor que la satisfaga. De esta manera, un cliente puede realizar peticiones a distintos servidores conformes y un servidor puede atender peticiones que provienen de clientes diversos incluso si son de distintos fabricantes. Esta característica posibilita a las Infraestructuras de Datos Espaciales (IDEs) la posibilidad de construir portales que permitan la visualización de los datos, consultado directamente las fuentes originales.

3.2.2 Interoperabilidad

Con respecto al geoprocesamiento, la interoperabilidad se refiere a la capacidad de los sistemas digitales de intercambiar libremente todas las clases de información espacial y controlar el software capaz de manejar esta información,

Con el web mapping, *[03_04]* como con muchas otras aplicaciones en la web, existe un gran número de servidores de múltiples empresas y organizaciones. La oportunidad que ofrece la web de un acceso amplio no se aprovecha si cada servidor tiene una implementación propietaria sin una especificación de interfaz pública. Incluso si la interfaz está documentada públicamente, puede que no sea estándar.

Para encaminar este problema hacia una solución, el OGC diseñó un web mapping no propietario basado en interfaces, codificaciones y esquemas abiertos. El Programa de Especificación y el Programa de Interoperabilidad del OGC proporcionan a la industria un proceso para planificar, desarrollar, revisar y adoptar las especificaciones OpenGIS para sus interfaces, codificaciones y esquemas. Estas especificaciones permiten a los servicios de geoprocesamiento, datos y aplicaciones que puedan interoperar entre sí.

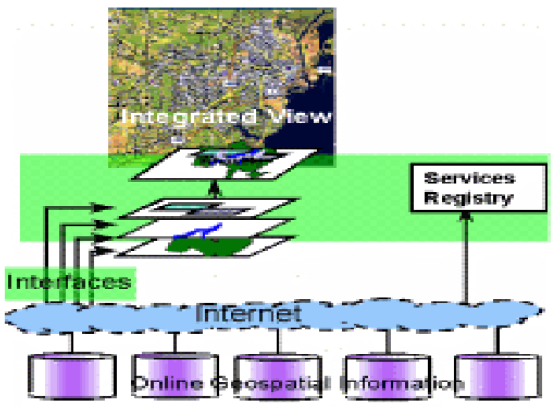

Figura 3.1. WMS – Superposición de mapas interoperables

Los fabricantes que componen el banco de pruebas de los estándares OGC están logrando esta interoperabilidad en su software (que se puedan superponer mapas procedentes de diferentes servidores de internet), gracias al conjunto de interfaces de la OpenGIS Implementation Specification (que incluyen la especificación WMS).

Así, un mismo cliente tiene acceso web a todos los servidores de mapas disponibles y múltiples fuentes de datos, accediendo a dichos servidores por la interfaz común. Este concepto de interoperabilidad se puede ver en la figura siguiente.

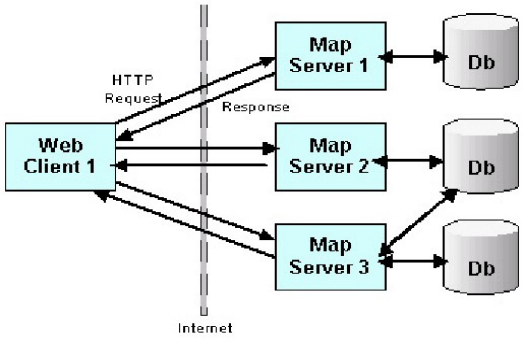

Figura 3.2. WMS – Interoperabilidad C/S

Este acercamiento permite, entre otras cosas, que el usuario ejecute un único cliente que acceda a todas las capacidades de cada servidor; lo que posibilita un entorno de aplicaciones más abierto donde las mejores características de los servicios web disponibles, de manera sencilla, se pueden combinar de nuevas formas para solucionar problemas cada vez más complejos.

3.2.3 La especificación WMS como una API

Una API, en el contexto que nos ocupa, es una definición de interfaz que permite escribir servicios OpenGIS para programas de aplicación sin conocer los detalles de cómo está implementada.

La especificación WMS es en esencia una API que permite a los programadores añadir una interfaz interoperable a diferentes sistemas de geoprocesamiento de distintos fabricantes y tipos (GIS, imágenes, navegación, ...).

Cualquier API en línea tiene tres componentes principales:

- Un vocabulario para solicitar información (request)
- Un vocabulario para responder las solicitudes (response)
- Un protocolo para el intercambio de solicitudes y respuestas

La especificación WMS define el vocabulario y la sintaxis de las operaciones y comandos que permiten a los servidores web y clientes comunicarse sobre el protocolo HTTP.

3.2.4 Operaciones de la interfaz WMS

La especificación WMS 1.1.1 estandariza tres operaciones (dos obligatorias y una opcional) mediante las cuales los clientes solicitan los mapas, y estandariza el modo en que los servidores describen sus datos. Además, la especificación WMS define un conjunto de parámetros de consulta y su comportamiento asociado.

Las tres operaciones (de petición) son las siguientes:

- `GetCapabilities` (obligatoria), que permite decirle a otros programas qué mapas puede producir y sobre qué mapas se puede solicitar una operación `GetFeatureInfo`. Con esta operación también se puede obtener una descripción de las diferentes capas, los sistemas de coordenadas que admite el servicio o los formatos de datos en que es capaz de generar la respuesta.
- `GetMap` (obligatoria), por la que el servidor produce un mapa (como una imagen, una serie de elementos gráficos, o como un conjunto de datos de carácter geográfico)
- `GetFeatureInfo` (opcional), que le permite responder a preguntas básicas sobre algún punto del mapa.

La especificación WMS estandariza o define una sintaxis para los localizadores de recursos URL's que invoquen cada una de ellas. También se define para la operación `GetCapabilities` una codificación XML para la respuesta.

Un cliente WMS (por ejemplo, un navegador estándar) puede pedir a un servidor WMS que haga estas cosas simplemente enviando las peticiones en forma de URL's. El contenido de estas URL's dependerá de cuál de las tres tareas se requiera. Todas las URL's, no obstante, incluyen el número de versión de la especificación WMS y un parámetro de tipo petición. Además,

- Para producir un mapa, los parámetros URL indican qué área de la Tierra va a ser mapeada, las coordenadas del sistema a usar, el tipo de información que queremos que se muestre, el formato de salida deseado, el tamaño de salida en píxels, el estilo de renderizado u otros parámetros.
- Para consultar el contenido de un mapa, los parámetros URL indican qué mapa está siendo consultado y qué localización del mapa es la que interesa.
- Para preguntar al servidor de mapas por los mapas que guarda, los parámetros URL incluyen el tipo de petición “capabilities”.

A continuación vamos a estudiar en más detalle las tres operaciones mencionadas.

3.2.4.1 `GetCapabilities`

La petición `GetCapabilities` devuelve una descripción del contenido de la información del servicio WMS y de los parámetros de petición que acepta. La respuesta a una petición

`GetCapabilities` es, por tanto, información sobre el servicio en sí mismo e información específica sobre los mapas disponibles en el servidor.

Para solicitar un mapa, un cliente WMS tiene que especificar las capas de información y los estilos asociados, así como el formato final que desea. Para ello, lo primero es que el cliente averigüe qué le puede pedir al servidor de mapas (encontrar las capacidades del servicio del servidor). Para encontrar qué capas proporciona el servidor WMS y qué proyecciones soporta, un cliente WMS hace una petición de capacidad (Capabilities Request). Otro propósito de esta petición es publicar los servicios GetMap que proporciona el servidor.

Un cliente invoca `GetCapabilities` para obtener la lista completa de interfaces que soporta un servidor WMS y cuáles capas de mapas puede servir dicho servidor en respuesta a una petición de `GetMap`. La lista también incluye información acerca de si el servidor soporta la operación opcional `GetFeatureInfo`.

A continuación mostramos toda la funcionalidad que proporciona la operación `GetCapabilities` a los clientes WMS de los servidores WMS:

- Todas las interfaces que puede soportar el servidor.
- Los formatos de imágenes (ráster) que puede servir: GIF, JPG, PNG, ...
- La lista de los sistemas de referencia espaciales disponibles para la entrega de mapas desde el servidor WMS.
- La lista de todas las excepciones y formatos que contempla el servidor WMS. La inclusión de valores para este atributo es opcional.
- La lista de todas las propiedades específicas que el vendedor proporciona para modificar o controlar las acciones de un servidor WMS particular, con el valor actual de cada una. La inclusión del valor para este atributo también es opcional.
- La lista de una o más capas de mapas disponibles en el servidor WMS particular. La inclusión de un valor para este atributo es opcional.
- Información de si el servidor soporta la interfaz opcional FeatureInfo.

La lista se devuelve como un documento XML que es válido respecto al DTD Capabilities. El DTD completo se incluye en el OGS Web Map Service Specification *[03_05]*, y especifica los contenidos obligatorios y opcionales de la respuesta y cómo se formatean los contenidos.

Un ejemplo de petición `GetCapabilities` podría ser el siguiente, que nos permite obtener las características del servidor de datos espaciales de Navarra, y que nos devuelve un fichero XML donde podemos ver las diferentes capas: mapa base, vías pecuarias, enclaves naturales, paisajes protegidos, zonificación de aves esteparias, etc.

http://idena.navarra.es/ogc/wms.aspx?REQUEST=GetCapabilities

3.2.4.2 `GetMap`

La petición `GetMap` devuelve una imagen de mapa cuyos parámetros geoespaciales y dimensiones están bien definidos. La operación la invoca un cliente para recuperar un conjunto rectangular de pixels, que contienen un dibujo de un mapa cubriendo una cierta zona geográfica o bien un conjunto de elementos gráficos que están en dicha área solicitada por el cliente. La operación también permite al cliente especificar diferentes capas, el sistema de referencia espacial (*Spatial Reference System*, SRS), el área geográfica, y otros parámetros que describen el formato del mapa devuelto. Una vez recibida la petición de `GetMap`, el servidor WMS bien puede satisfacer la petición o lanzar una excepción acorde con las instrucciones de excepción contenidas en la petición `GetMap`.

El servidor WMS tiene que ser capaz de entregar un mapa vía HTTP una vez que recibe una petición de un cliente WMS, como podría ser la siguiente instrucción:

```
http://www.airesip.org/wms/process.cgi?REQUEST=GetMap&
FORMAT=image/gif&WIDTH=640&HEIGHT=480&LAYERS=temperature
&SRS=EPSG:4326&BBOX=-110.,40.,-80.,30.&&VERSION=1.1.0
```

En la instrucción de ejemplo previa, `http://www.airesip.org/` es el hostname del servidor, `wms/` es su ruta, y `/process.cgi` el nombre del script CGI que procesa las peticiones de los clientes WMS. Un interrogante se añade detrás del nombre del script para separarlo de la lista de parámetros, y cada parámetro de la lista va separado por un ampersand (&). Los parámetros pueden aparecer en cualquier orden, y no son sensibles a mayúsculas. Por ejemplo, se espera obtener una imagen en formato .gif de 640x480, de la capa *temperature*, y sobre la zona geográfica que define el parámetro BBOX.

La operación `GetMap` permite que los clientes WMS pueden construir mapas personalizados. Por ejemplo, si dos o más mapas se producen con una petición en donde el valor de los parámetros BBOX (Bounding Box), SRS (Spatial Reference System) y tamaño de salida son iguales, los resultados se pueden superponer con precisión para componer un nuevo mapa. Además, el uso de formatos de imagen que soportan transparencias permite a las capas más bajas quedar visibles.

A continuación mostramos una lista con los parámetros que puede especificar un cliente en una petición `GetMap`, indicando si son obligatorios (R) u opcionales (O):

- *`VERSION=version`* (R): versión de la petición
- *`REQUEST=GetMap`* (R): nombre de la petición
- *`LAYERS=layer_list`* (R): lista de capas de mapas separadas por comas
- *`STYLES=style_list`* (R): lista estilos de renderizado para cada capa solicitada en la lista de capas *LAYERS*

- *SRS=namespace:identifier* (R): SRS (Sistema de Referencia Espacial)
- *BBOX=minx,miny,maxx,maxy* (R): esquinas de la caja personalizada (inferior izquierda y superior derecha), en unidades SRS
- *WIDTH=output_width* (R): anchura en píxeles de la imagen de mapa
- *HEIGHT=output_height* (R): altura en píxeles de la imagen de mapa
- *FORMAT=output_format* (R): formato de salida de la imagen de mapa
- *TRANSPARENT=TRUE/FALSE* (O): transparencia del fondo del mapa (por defecto está a FALSE)
- *BGCOLOR=color_value* (O): color rojo-verde-azul en hexadecimal para el color de fondo (por defecto, 0xFFFFFF).
- *TIME=time* (O): valor de tiempo para la capa deseada
- *ELEVATION=elevation* (O): elevación de la capa deseada
- Parámetros específicos del vendedor (O)
- *SLD=styled-layer_descriptor_URL* (O): en el caso de que el WMS soporte la especificación SLD, indicará su URL.
- *WFS=web_feature_service_URL* (O): en el caso de que el WMS sea del tipo SLD y use WFS, será la URL del WFS que proporciona las características que se pueden representar mediante símbolos utilizando SLD.

Una vez que conocemos todos los parámetros que se pueden especificar en una petición `GetMap`, mostramos ahora una petición de este tipo lanzada contra el servidor WMS del ICC, concretamente contra el servicio web WMS de la Ortofoto de Catalunya 1:5 000, que se compone de una única capa de información (Ortofoto_5000) *[03_07]*:

> *http://galileo.icc.es/wms/servlet/icc_orto5m_r_r?REQUEST=GetMap&VERSION=1.1.1&SERVICE=WMS&SRS=EPSG:23031&LAYERS=Ortofoto_5000&STYLES=_&FORMAT=JPEG&BGCOLOR=0xFFFFFF&TRANSPARENT=TRUE&EXCEPTION=INIMAGE&BBOX=514128.35,4678107.64,516047.93,4680027.22&WIDTH=520&HEIGHT=520*

Si ejecutamos la petición en un navegador web, se nos cargará una página con la siguiente imagen en formato JPEG:

Figura 3.3. WMS – Salida petición GetMap contra WMS del ICC

3.2.4.3 `GetFeatureInfo`

La petición `GetFeatureInfo` devuelve información acerca de características particulares mostradas en un mapa. El uso típico de esta operación es que un usuario visualice un mapa devuelto de una petición `GetMap` y escoja un punto de este mapa sobre el cuál desea obtener más información. Así, si un servidor WMS soporta esta operación, sus mapas se dice que son "consultables", ya que el cliente puede solicitar información adicional sobre características del mapa especificando nuevos parámetros (como el desplazamiento X e Y desde la esquina superior izquierda donde está el punto en cuestión, la capa que se desea investigar o el formato en que se desea que se devuelva la información).

El mapa se identifica incluyendo toda la información contenida en la petición `GetMap`. De hecho, si la petición `GetMap` previa no se repite correctamente en la petición GetFeatureInfo, los resultados no estarán definidos y se producirá una excepción.

En respuesta a la petición `GetFeatureInfo`, el servidor WMS puede devolver una página de texto/HTML/XML, una imagen o incluso un documento de texto.

Los parámetros de la petición `GetFeatureInfo` son los siguientes:

- *VERSION=version* (R): versión de la petición
- *REQUEST=GetFeatureInfo* *<map_request_copy>* (R): copia parcial de los parámetros de la petición GetMap que generaron el mapa sobre el que deseamos obtener más información.
- *QUERY_LAYERS=layer_list* (R): lista de una o más capas consultadas separadas por comas
- *INFO_FORMAT=output_format* (O): formato de respuesta de la información
- *FEATURE_COUNT=number* (O): número de características sobre las cuales retornar información (por defecto, 1)
- *X=pixel_column* (R): coordenada X en píxeles de la característica (medidas desde la esquina superior izquierda, cuyo valor X=0).
- *Y=pixel_row* (R): coordenada Y en píxeles de la característica
- *EXCEPTIONS=exceptions_format* (O): el formato en que el servidor WMS debe informar sobre las posibles excepciones
- Parámetros específicos del vendedor (O)

3.3 Web Feature Service

3.3.1 Descripción

Un WFS *[03_08]* permite a un cliente realizar operaciones de consulta y, en su caso, manipulación de datos, en un conjunto de características geográficas, usando HTTP como la plataforma de computación distribuida. Un usuario o servicio puede combinar, usar y administrar geodatos (información característica de un mapa) de diferentes fuentes invocando las siguientes operaciones WFS de manipulación de datos:

- Obtener o consultar entidades geométricas (*features*) basadas en restricciones espaciales y no espaciales
- Crear una nueva entidad geométrica
- Borrar una entidad geométrica
- Actualizar una entidad geométrica
- Bloquear una entidad geométrica

Nota: a lo largo del documento, se adopta el término "entidades/objetos geográficos" a lo que en inglés se denomina "*features*", y que son los elementos geográficos básicos de un modelo vectorial.

Del mismo modo que el OGC Web Map Service *[03_09]* permitía a un cliente superponer imágenes de mapa para mostrarlos y que procedían de múltiples servidores WMS de Internet, el OGC Web

Feature Service permite a un cliente recuperar y actualizar datos geoespaciales codificados en GML de múltiples servidores WFS. Es decir, el primero nos permitía trabajar con vistas ráster y superponerlas, mientras que el WFS nos permite hacer lo mismo con mapas vectoriales (que además ofrecen más posibilidades de consulta).

Lo que se descarga del sistema es en realidad un fichero GML *[03_10]*, una extensión del formato XML para transportar y almacenar información geográfica, incluyendo la geometría y las propiedades de los objetos geográficos. Un fichero GML contiene entre otras cosas las coordenadas y los atributos recortados de la zona especificada, lo que el servidor de mapas cliente transforma de nuevo en un mapa.

3.3.2 Arquitectura WFS

A continuación se adjunta una figura que muestra los componentes necesarios para servir características geográficas y procesar peticiones transaccionales de clientes utilizando el protocolo HTTP como entorno de computación distribuida.

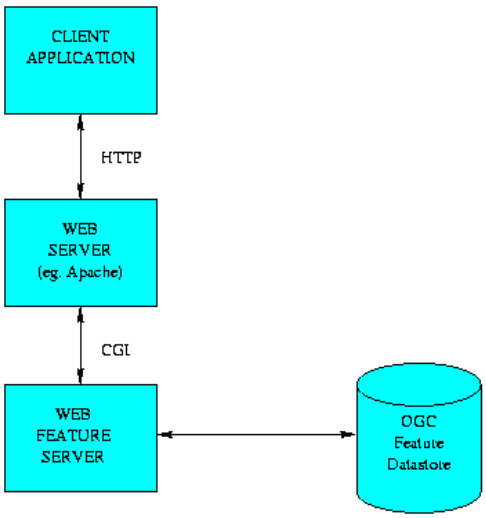

Figura 3.4. WFS – Arquitectura

- Aplicación Cliente. Será cualquier programa o proceso que se comunique con el servidor web usando HTTP. Lo más normal es que se trate de un navegador web, aunque puede tratarse de otro tipo de programas (por ejemplo un editor de geodatos que se comunique con un servidor HTTP).
- Servidor Web. Cualquier programa que atienda peticiones HTTP (por ejemplo, Apache).
- Servidor Web Feature. Programa o módulo que implementa las interfaces que proporcionan comportamiento transaccional y/o permiten las operaciones de consulta sobre los elementos geométricos
- Almacén de Datos de Objetos Geográficos. Componente software para almacenar y administrar las propiedades espaciales y no espaciales de los objetos geográficos. Puede ser una base de datos relacional SQL con extensión para soportar entidades geográficas (geobase de datos como PostGIS, SDE o Oracle Spatial), una base de datos ARCINFO, un almacén de datos basado en XML, etc ..

3.3.3 Procesamiento de peticiones WFS

En el procesamiento de peticiones Web Feature Service, el protocolo que se sigue es el siguiente:

- La aplicación cliente solicita el documento de capacidad (*capabilities*) del servicio. Este documente describe todas las operaciones que soporta el servicio WFS y una lista de todos los tipos de características que puede servir.
- Una aplicación cliente (opcionalmente) realiza una petición al servidor para obtener la definición de una o más de los tipos de elementos que puede servir.
- Basándose en la definición de los tipos de elementos, la aplicación cliente genera una petición como se especifica en el documento OGC Web Feature Server Specification.
- La petición se envía a un servidor web.
- El WFS se invoca para leer y servir la petición.
- Cuando el WFS ha complatado de procesar la petición, generará un informe de estado y lo devolverá al cliente. Si ocurre algún error, el informe de estado lo indicará.

La figura 4.5 muestra un diagrama simplificado del protocolo de paso de mensajes entre una aplicación cliente, un servidor WFS y un almacén de datos (que se usa para el almacenamiento persistente de la geometría):

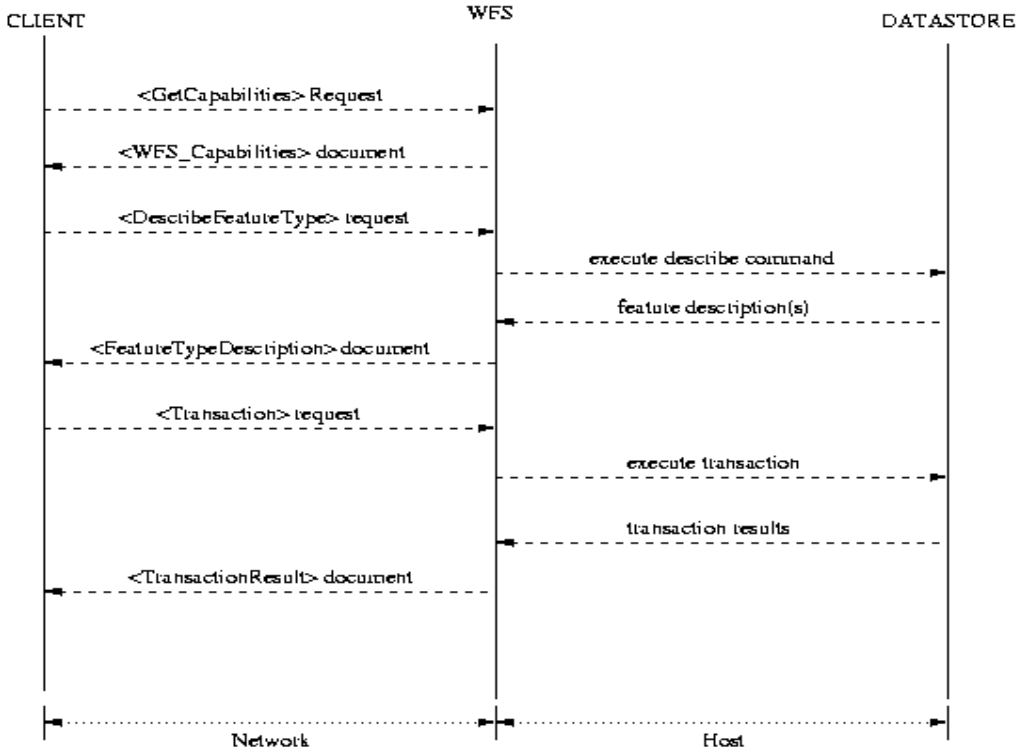

Figura 3.5. WFS – Paso de mensajes de una petición de transacción típica

3.3.4 Operaciones de la interfaz WFS

Para dar soporte transaccional y procesamiento de consultas, se definen las siguientes operaciones WFS:

3.3.4.1 GetCapabilities

Un servidor web de características debe ser capaz de describir sus capacidades. Específicamente, debe indicar qué tipos de objetos geográficos puede servir, y qué operaciones soporta cada tipo de objeto o elemento.

3.3.4.2 DescribeFeatureType

Un servicio web de características tiene que ser capaz, una vez se le realiza una petición, de describir la estructura de cualquier tipo de objeto geográfico que pueda servir.

3.3.4.3 GetFeature

Un WFS tiene que poder servir una petición de recuperación de instancias de objetos geográficos. Además, el cliente tiene que poder especificar qué propiedades de los objetos quiere recuperar.

3.3.4.4 GetGmlObject

Es una nueva operación que añade la especificación WFS 1.1.0, junto con el servicio XLink. GetGmlObject proporciona la capacidad de situar referencias dentro de las propias definiciones de referencias de la geometría, interna o externamente a los elementos de datos GML. Esto permite la separación entre características geográficas y atributos.

3.3.4.5 Transaction

Un WFS debe poder servir peticiones de transacción. Una petición transaccional se compone de operaciones que modifican elementos geográficos, es decir, operaciones de creación, actualización y borrado sobre dichos elementos.

3.3.4.6 LockFeature

Un servicio web de objetos geográficos debe poder procesar una petición de bloque en una o más

instancias de un tipo de elemento geográfico durante la duración de una transacción. Esto asegura el soporte para las transacciones seriales (serializables).

3.3.5 Tipos de servicios WFS

Basándonos en la descripción de las operaciones del apartado previo, podemos definir tres clases de servicios web de características:

3.3.5.1 WFS básico

Un WFS básico implementará las operaciones `GetCapabilities`, `DescribeFeatureType` y `GetFeature`. Por tanto, se puede considerar como un servicio web de características de solo lectura.

3.3.5.2 WFS XLink

XLink es el lenguaje, definido en términos de marcas XML, que nos va a permitir introducir enlaces en nuestros ficheros XML, de modo que podamos relacionar unos ficheros con otros. Un XLink permite establecer una relación entre dos o más recursos en la web, sin que necesariamente estos recursos sepan que están enlazados.

Un WFS XLink soportará todas las operaciones de un WFS básico y además implementará la operación `GetGmlObject` para XLinks remotos y/o locales, y ofrecerá la opción de que la operación `GetGmlObject` se realice mientras se ejecutan operaciones `GetFeature`.

3.3.5.3 WFS transaccional

Un WFS de transacciones podrá ofrecer todas las operaciones de un servicio web de características básico y además implementará la operación `Transaction`. Opcionalmente, un WFS transaccional podrá implementar las operaciones `GetGmlObject` y/o `LockFeature`, explicadas en el apartado 4.3.4.

3.4 Ejemplos de servicios OGC

Podemos encontrar más ejemplos en la red de servicios WMS y WFS. A continuación mostramos una

lista:

- Atlas virtual de aves terrestres de España
  - Capabilities: *http://161.111.161.171/cgi-bin/AtlasAves.exe?service=WMS&request=GetCapabilities*
  - Resumen: mapas de distribución y listados de especies de la avifauna terrestre española. Utilizan como aplicación servidora de mapas MapServer, un programa CGI creado por la universidad de Minnesota, y almacenan los datos temáticos en una geodatabase PostgreSQL/PostGIS.
- Servidor ráster del Instituto Cartográfico de Catalunya
  - Capabilities: *http://shagrat.icc.es/lizardtech/iserv/ows?REQUEST=Getcapabilities&VERSION=1.1.1*
  - Resumen: geoservicios del Instituto Cartográfico de Catalunya. Poseen un servidor que se compone de un sólo servicio con múltiples capas, donde cada capa se corresponde con un mapa continuo diferente. Actualmente disponen de 7 mapas de Catalunya (topográficos, geológicos y ortofotos).
- IDENA – Infraestructura de datos catastrales de Navarra
  - Capabilities: *http://idena.navarra.es/ogc/wms.aspx?REQUEST=GetCapabilities&SERVICE=wms*
  - Resumen: ofrece un servicio estandarizado para que la información geográfica de Navarra y de otros ámbitos, sea quien sea su productor, pueda ser utilizada por los usuarios.
- IDERIOJA – Gobierno de la Rioja
  - Capabilities: *http://wms.larioja.org/request.asp?request=GetCapabilities*
  - Resumen: OGC WMS del Gobierno de la Rioja
- NASA SVS Image Server
  - Capabilities: *http://aes.gsfc.nasa.gov/cgi-bin/wms?%20SERVICE=WMS&VERSION=1.1.0&REQUEST=GetCapabilities*
  - Resumen: Web Map Server mantenido por el estudio de visualización científica de la NASA
- Tsunami Disaster Data
  - Capabilities: *http://www.mapsherpa.com/cgi-bin/wms_iodra?SERVICE=WMS&REQUEST=GetCapabilities*
  - Resumen: Servidor WMS que contiene datos sobre desastres provocados por tsunamis. Contiene mapas de diferentes países afectados, incluye una herramienta para identificar características de los mapas que permite obtener más información y estadísticas, ...
- World Mineral Deposits
  - Capabilities: *http://apps1.gdr.nrcan.gc.ca/cgi-bin/worldmin-en-ca_ows?request=GetCapabilities*
  - Resumen: mapas de depósitos minerales compilados por la Geological Survey de Canadá durante tres décadas

Podemos encontrar herramientas en la red que nos facilitan la tarea de conectarnos a un WMS/WFS determinado y trabajar con él, construyendo por nosotros las peticiones sobre estos servicios web. Estas aplicaciones clientes WMS/WFS nos permiten descubrir las capas disponibles en un servicio determinado, cargar capas de orígenes diversos, y una funcionalidad mínima: zoom, desplazamiento, etc .

Un visor de este tipo bastante sencillo (requiere .NET 2.0) es *Gaia* (cuyo instalable y documentación podemos descargar desde la página web http://www.thecarbonportal.net/modules.php).

También podemos encontrar visores en línea como Humboldt (http://humboldt-viewer.sourceforge.net/demo/index.hml).

4 Google Maps

4.1 Introducción a Google Maps y a la tecnología AJAX

4.1.1 Google Maps

Google Maps *[04_01]* es el nombre de un servicio de mapas web gratuito de Google. Es un servidor de aplicaciones de mapas en web proporcionado en http://maps.google.es. Ofrece imágenes de mapas desplazables, así como fotos satelitales del mundo entero e incluso la ruta entre diferentes ubicaciones. Además, permite incluir el visor de mapas en otras webs por medio de la API de Google Maps y personalizarlo gracias a dicha API.

Un producto relacionado es Google Earth, una aplicación multiplataforma que ofrece vistas de la Tierra en 3D y algunas características mejoradas, pero que al tratarse de una aplicación de escritorio no es posible integrarlo en páginas web.

Google Maps permite que los usuarios naveguen por los mapas arrastrando el ratón (o mediante las flechas de dirección), o que usen la rueda del ratón para disminuir o aumentar el zoom (o bien las teclas "-" y "+"). Los usuarios pueden introducir una dirección, una intersección o un área en general para buscar en el mapa.

Los resultados de las búsquedas se pueden restringir a un contexto o área determinado. Así, por ejemplo, un usuario puede introducir una consulta como "Bodega en Córdoba" para buscar bodegas en esta ciudad. Incluso puede restringir aún más la búsqueda introduciendo *category: Elaboracion De Vino* para buscar bodegas donde produzcan vino en Córdoba. Lo mismo se puede hacer para localizar una amplia variedad de negocios.

Como otros servicios de mapa, Google Maps permite generar rutas entre dos puntos, en base a las direcciones y sentido de la circulación. Así, proporciona al usuario una lista paso a paso de cómo llegar desde el punto origen al destino, junto con una estimación del tiempo necesario para alcanzar el destino.

Google Maps ofrece tres modos de vistas por defecto:

- Mapa, que son vistas de mapa de las calles o las carreteras (en el nivel de zoom correspondiente)
- Satélite, que son fotografías satelitales y aéreas de alta resolución (especialmente en las grandes ciudades y en países como Holanda o Suiza, en los que la resolución es submétrica;

en otras zonas del planeta la resolución es más pobre)
- Híbrido, donde se muestran los mapas de las calles superpuestos sobre las vistas satelitales y aéreas de alta resolución

Google también proporciona una aplicación en línea (http://www.google.com/uds/solutions/wizards/mapsearch.html), en la cual un usuario puede buscar la localización que desee en el mapa (bien arrastrándolo o introduciendo la dirección), personalizar las dimensiones que desea que tenga de salida, el nivel de zoom, ..., y pulsar el botón de "Generar Código", de forma que la aplicación genera un trozo de código para que directamente se incluya en una página web (copiando y pegando). Los mapas que genera la aplicación guiada sólo muestran una marca simple, y el usuario no tiene control sobre los contenidos.

En cuanto a la tecnología, Google Maps está basado en JavaScript. Según el nivel de zoom, un mapa podría descomponerse en decenas de miles de cuadrados. Esta descomposición se realiza del lado del servidor, de forma que cada cuadrado se almacena en un fichero cuyo nombre indica su longitud, su latitud y su nivel de zoom. Así, conforme el usuario arrastra el mapa, cada cuadrado de la malla se descarga individualmente del servidor, mostrándose al usuario el mapa completo. Es por ello que Google Maps monta las imágenes tan rápido, porque lo hace por cuadrículas, así que cuando el usuario desplaza el mapa sólo se añaden las cuadrículas adyacentes en vez de cargar toda la imagen completa. Cuando un usuario busca una localización, en la vista resultado se marca el punto buscado con una imagen PNG. La técnica que se usa para proporcionar una alta interactividad realizando peticiones asíncronas con JavaScript y XML se conoce como AJAX. De hecho, Google Maps fue desarrollado utilizándose el framework AjaXSLT.

Existen otros productos similares a Google Maps, como por ejemplo Yahoo! Maps, si lo que se quiere es una forma sencilla de incluir un mapa en una web sin tener que programar. Yahoo! Maps (http://developer.yahoo.net/maps/) proporciona un servicio sencillo que permite al usuario pasar en una lista XML los puntos y obtener un mapa. Se le pueden pasar iconos personalizados, links html, y texto descriptivo para cada punto en el XML.

La API de Google Maps requiere que el usuario se involucre en la implementación. Google Maps, después de todo, es una API, no una aplicación final. La ventaja es que el usuario tiene muchísimo más control de su aplicación. Se encuentra ante un modelo de eventos en el cuál puede reaccionar a cada click en el mapa, a cada zoom, a cada movimiento. El control sobre los iconos es mucho más personalizado (la imagen, las sombras, ...). Pero lo más importante es que el usuario obtiene un componente de mapa que se puede encajar sin más en su página web.

4.1.2 Breve historia de Google Maps

La versión beta de Google Maps salió el 7 de febrero de 2005. Su interfaz creó sensación (por el hecho de poder arrastrar los mapas). Hasta entonces no se tenía constancia de que una aplicación web pudiera ser tan interactiva y responder tan rápido, sin necesidad de recargar la página. Además, rompió con todas las reglas, ya que esta interactividad no implicaba tener que utilizar plugins de Flash o tecnologías similares, sino que Google tan sólo se basó en las herramientas de que disponen los navegadores: JavaScript, CSS y la función XmlHttpRequest.

El 18 de febrero de 2005, Jesse James Garrett publicó un artículo semanal que le dio nombre a este nuevo estilo de desarrollo web: "Ajax: un nueva aproximación a las aplicaciones web". De repente, Google Maps no era sólo una aplicación de mapas revolucionaria, sino que se convirtió en el tipo de aplicación web que cualquier programador quería desarrollar. De hecho, Tim O'Reilly, fundador de O'Reilly Media, acuñó otra frase, "Web 2.0", que sirvió para diferenciar aún más entre el comportamiento típico de las aplicaciones web de siempre y el comportamiento interactivo sin necesidad de recargar páginas de Google Maps.

El 29 de junio de 2005, Google lanzó la versión 1 de su API. Esto permitió al usuario pasar de ser un simple consumidor de mapas a convertirse en productor, ya que con la API cualquiera puede añadirla a su página web y modificar la interfaz de Google Maps a su gusto. Con la contraseña oficial de desarrollador, la API es libre de usarse para cualquier sitio web (siempre para uso no comercial y accesible públicamente).

El 22 de julio de 2005, Google lanzó una vista dual de su Google Maps. Esta vista combina los mapas vectoriales y la vista satelital en una nueva vista, llamada "Híbrida", donde aparecen los nombres de las calles y algunas ilustraciones sobre imágenes del mundo real. Esto hace más fácil encontrar rutas entre dos puntos.

El 3 de abril de 2006, Google lanzó la versión 2 de su API. Esta nueva versión introdujo muchas nuevas características interesantes, como aumentar los niveles de zoom, controles de mapa adicionales, ..., pero también rompió la compatibilidad con la versión anterior.

El 11 de junio de 2006, Google añade funciones de geocodificación (geocoding) a la API. La geocodificación es el proceso de traducir direcciones en coordenadas geográficas. Así se pueden situar marcas en las direcciones que desea el usuario. Se puede acceder al geocodificador de la API de Google Maps por medio de peticiones HTTP o directamente desde dentro por JavaScript.

Empezando el 28 de febrero de 2007, Google lanza "Google Traffic Info", que incluye el estado del tráfico en tiempo real en los mapas de las 30 ciudades más importantes de los Estados Unidos.

El 5 de abril de 2007 Google extiende la funcionalidad de Google Maps con "Mis mapas", una nueva herramienta que permite personalizar los mapas de forma que cualquier usuario puede marcar sus lugares favoritos en el mapa, dibujar líneas y formas para resaltar rutas y áreas concretas, añadir sus propios textos, fotos y vídeos, y publicar su mapa en la Red.

4.1.3 Proyecciones y Google Maps

Cuando tratamos de representar datos terrestres (3D) en un mapa (2D) nos encontramos con dificultades. En un mapa 2D representaremos imágenes que estarán geoespacialmente referenciadas, esto es, contendrán información geográfica referente a cada punto de la imagen.

Al hecho de trasladar la Tierra en 3D a una superficie plana es a lo que llamamos proyección. La proyección trata de compensar la carencia de una tercera dimensión mediante diferentes técnicas, pero de una u otra forma introduce cierto tipo de distorsión. Hay cuatro tipos de distorsión posibles: distancia, dirección, forma y área. Ningún tipo de proyección puede minimizar los cuatro tipos de distorsión, por lo que cada tipo de proyección se afana en minimizar un tipo diferente de distorsión.
Debido a que existen multitud de proyecciones, un buen SIG debe manejar estos conceptos. Ésta es una de las principales razones que hacen que un SIG sea un sistema complejo. En el caso del SIG que nos ocupa, Google Maps utiliza coordenadas polares (latitud/longitud) y no maneja proyecciones actualmente. Por esta razón y otras, como el hecho de que Google Maps no permite realizar análisis espacial a partir de sus datos de base y otros que pueda introducir el usuario, cabría preguntarse si Google Maps es realmente un SIG.

Parece obvio pues que Google Maps es más bien una aplicación de los SIG que un SIG. De hecho, para el usuario Google Maps es un cliente de visualización cartográfica programado en JavaScript. Debido a las limitaciones de cálculo que tiene JavaScript, al interpretarse en el navegador, Google Maps es muy buen visor de mapas, pero no podría constituir la base de un SIG corporativo. Google Maps, pese a no ser un SIG, ha contribuido a la eclosión social de los SIG, un mundo anteriormente reducido para algunas grandes empresas.

4.1.4 AJAX

AJAX (***A**synchronous **J**avaScript **A**nd **XML***), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente y mantienen comunicación asíncrona con el servidor en segundo plano por medio de una capa adicional intermediaria (el motor de AJAX). De

esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla, lo que incrementa la interactividad, velocidad y usabilidad en la aplicación web.

AJAX no es una tecnología, sino varias, ya que incorpora XHTML y CSS para la presentación basada en estándares; exhibición e interacción dinámicas usando el DOM; XSLT y XML para el intercambio y manipulación de datos; recuperación de datos asíncrona usando XMLHttpRequest; y JavaScript, que interconecta todo lo demás.

En resumen, el concepto en que se basa AJAX es cargar y componer una página, luego mantenerse en esa página mientras scripts y rutinas van al servidor buscando, en *background* (en segundo plano), los datos que son usados para actualizar esa página sólo re-renderizando la página y mostrando u ocultando porciones de la misma. En la figura siguiente mostramos la diferencia entre el modelo clásico de aplicación web y el modelo AJAX de aplicación web (esta imagen ha sido capturada del artículo original que publicó Jesse James Garret, *[04_02]*).

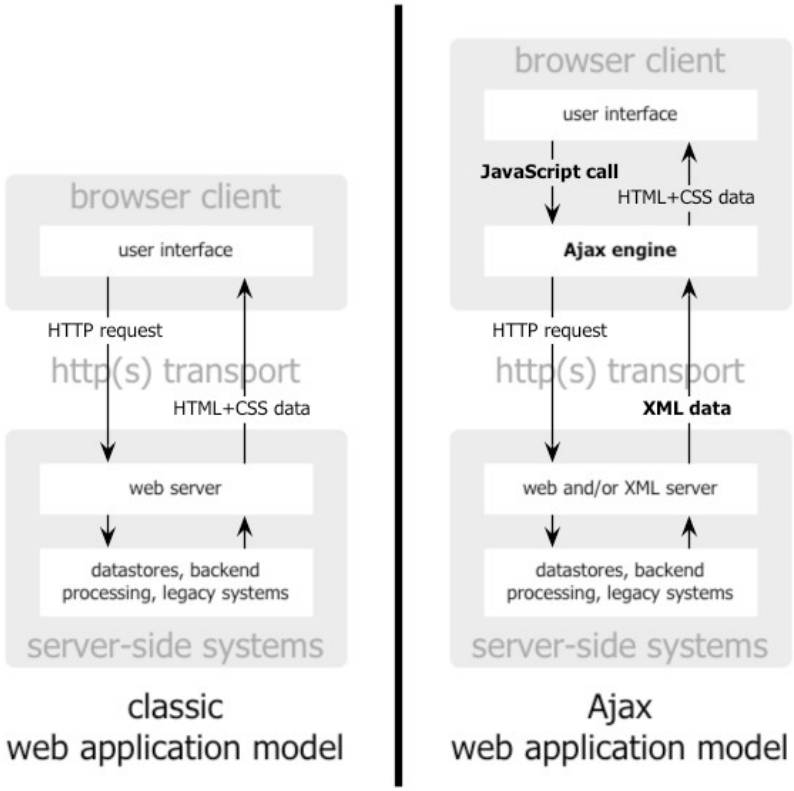

Figura 4.1. Comparación entre el modelo clásico de aplicación web y el modelo basado en AJAX

Como se comentó en el primer párrafo, AJAX sigue una arquitectura Cliente – Servidor de "cliente de nueva generación" (ofrece la sensibilidad de la interfaz de usuario de una aplicación de escritorio gracias a la naturaleza asíncrona de la ruta de comunicación multicanal de AJAX).

Los AJAX *toolkits* son colecciones de herramientas para entornos AJAX, cuyo objetivo es facilitar la tarea de desarrollar aplicaciones con AJAX. Pueden ir desde librerías de código que permitan manejar las peticiones y respuestas, los datos, …, etc, a IDEs como GWT (Google Web Toolkit), que permite escribir el código de las aplicaciones en Java para más tarde compilarlas a JavaScript y HTML, siendo el código resultante compatible con cualquier navegador web.

4.2 La API de Google Maps

4.2.1 Introducción: un mapa simple

Veamos a continuación la aplicación más simple posible de Google Maps. Antes de nada, hemos de obtener una clave de registro para poder utilizar la API en *http://www.google.com/apis/maps/signup.html*. Esta clave será válida en nuestro caso para todas las URLs que cuelguen del directorio *http://www.jlolmo.com/Google_Maps/*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ejemplo API Google Maps</title>
    <script
src="http://maps.google.com/maps?file=api&v=2&key=ABQIAAAAeQfmR5so439PA2uumrGf4h
SW75mperdLr8D3y5HZp15c3fT4XRThSm2vYELVLU04SW7g0sMAOleU2A"
             type="text/javascript"></script>
  </head>
  <body>
    <div id="map" style="width: 1300px; height: 600px"></div>
    <script type="text/javascript">
         var map = new GMap2(document.getElementById("map"));
         //zoom levels 0-17+, 0 == world
         map.setCenter(new GLatLng(37.4419, -122.1419), 15);
    </script>
  </body>
```

El ejemplo anterior muestra un mapa simple de 1300x600 centrado en Palo Alto, California:

- La url *http://maps.google.com/maps?file=api&v=2* incluida en la marca <script> es la localización donde se encuentra la librería Google Maps. Aquí es donde ponemos la clave.
- La etiqueta es donde situamos el mapa. Lo lógico es que la identifiquemos como *map* (`id="map"`), para así poder referenciarla desde la hoja de estilos CSS y establecer sus dimensiones (no obstante, en el ejemplo las indicamos en el atributo style).
- new GMap2() crea el mapa. En el constructor del objeto GMap2, le pasamos utilizando

DOM.

- map.setCenter() centra el mapa en un punto concreto (identificado por su latitud y longitud). Además, hace zoom en el nivel especificado. Google utiliza una escala de 17 niveles, donde el nivel 0 mostrará una vista satelital de la Tierra y el 17 llega al nivel de las calles.

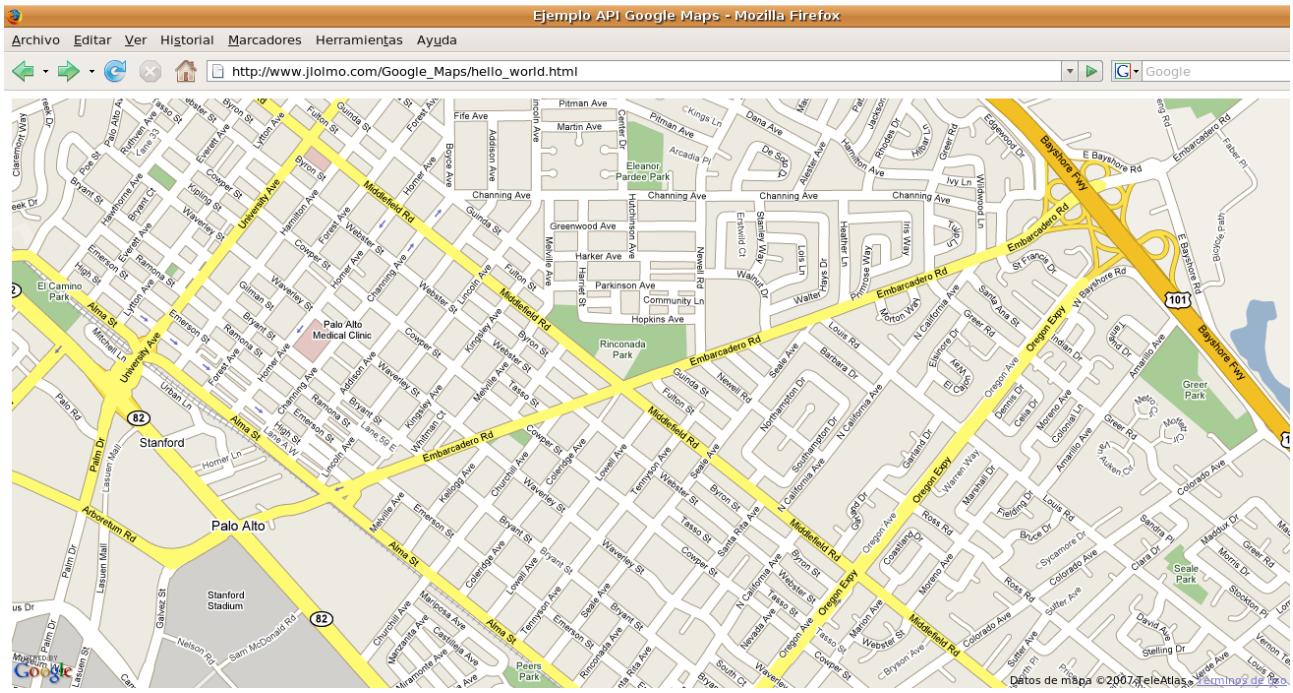

Figura 4.2. Ejemplo básico de la API de Google Maps

La API de Google Maps se compone de 20 objetos básicos *[04_03]*. Estos objetos entran dentro de 5 categorías: núcleo, controles de mapa, datos de usuario, eventos y AJAX. En los siguientes subapartados describiremos los objetos de cada categoría.

4.2.2 Objetos de núcleo

4.2.2.1 `GMap2`

Un objeto GMap2 es el mapa en sí. Podemos tener tantos objetos GMap2 como queramos en nuestra página. El constructor del objeto GMap2 requiere un único argumento, el contendor (container), que no es más que una etiqueta div. El atributo id de esta etiqueta es el único nombre del mapa. Por defecto, para determinar el tamaño del mapa, GMap2 usa el tamaño especificado en el atributo style de la etiqueta div.

Si queremos tener más de un mapa, simplemente hemos de tener varias etiquetas div, cada una con su

id único, e instanciar un objeto GMap2 en cada una. Por ejemplo:

```
<div id="overviewMap" style="width: 200px; height: 125px"></div>
<div id="detailMap" style="width: 800px; height: 500px"></div>
<script type="text/javascript">
  var overviewMap = new GMap2(document.getElementById("overviewMap"));
  var detailMap = new GMap2(document.getElementById("detailMap"));
</script>
```

Nota: el ejemplo anterior no podría ser renderizado, ya que se necesita un punto para centrar el mapa y el nivel de zoom.

4.2.2.2 GLatLng

Un objeto GLatLng es una coordenada geográfica (un punto con una latitud y una longitud). A continuación, seguimos con el ejemplo del apartado anterior y usamos el método map.setCenter() para centrar los mapas inicializados con GMap2 en el punto GLatLng:

```
<div id="overviewMap" style="width: 200px; height: 125px"></div>
<div id="detailMap" style="width: 800px; height: 500px"></div>
<script type="text/javascript">
  var overviewMap = new GMap2(document.getElementById("overviewMap"));
  var detailMap = new Gmap2(document.getElementById("detailMap"));

  //NOTE: This is the geographic center of the US
  var usCenterPoint = new GLatLng(39.833333, -98.583333);

  overviewMap.setCenter(usCenterPoint, 1);
  detailMap.setCenter(usCenterPoint, 7);
</script>
```

En la figura 5.3 podemos ver el resultado obtenido.

4.2.2.3 GLatLngBounds

Un objeto GLatLngBounds es una Bounding Box, que ya explicamos en el capítulo correspondiente al WMS. Como ya dijimos, una BBox representa el área geográfica de nuestro mapa.

Un objeto GLatLngBounds es un array de dos elementos GLatLng. El primer punto GLatLng representa la esquina inferior izquierda del mapa, y el segundo la esquina superior derecha.
El tamaño físico del mapa no cambia, y está definido en el atributo style de la etiqueta div. Pero los límites geográficos del mapa cambian constantemente, cada vez que lo arrastramos o movemos. Incluso aunque el punto en el que lo centramos siga siendo el mismo, si hacemos zoom, los límites de nuestra *Bounding Box* cambian. Para hallar los límites actuales podemos usar los siguientes métodos:

- map.getCenter(), que devuelve un objeto GLatLng
- map.getBounds(), que devuelve un objeto GlatLngBounds

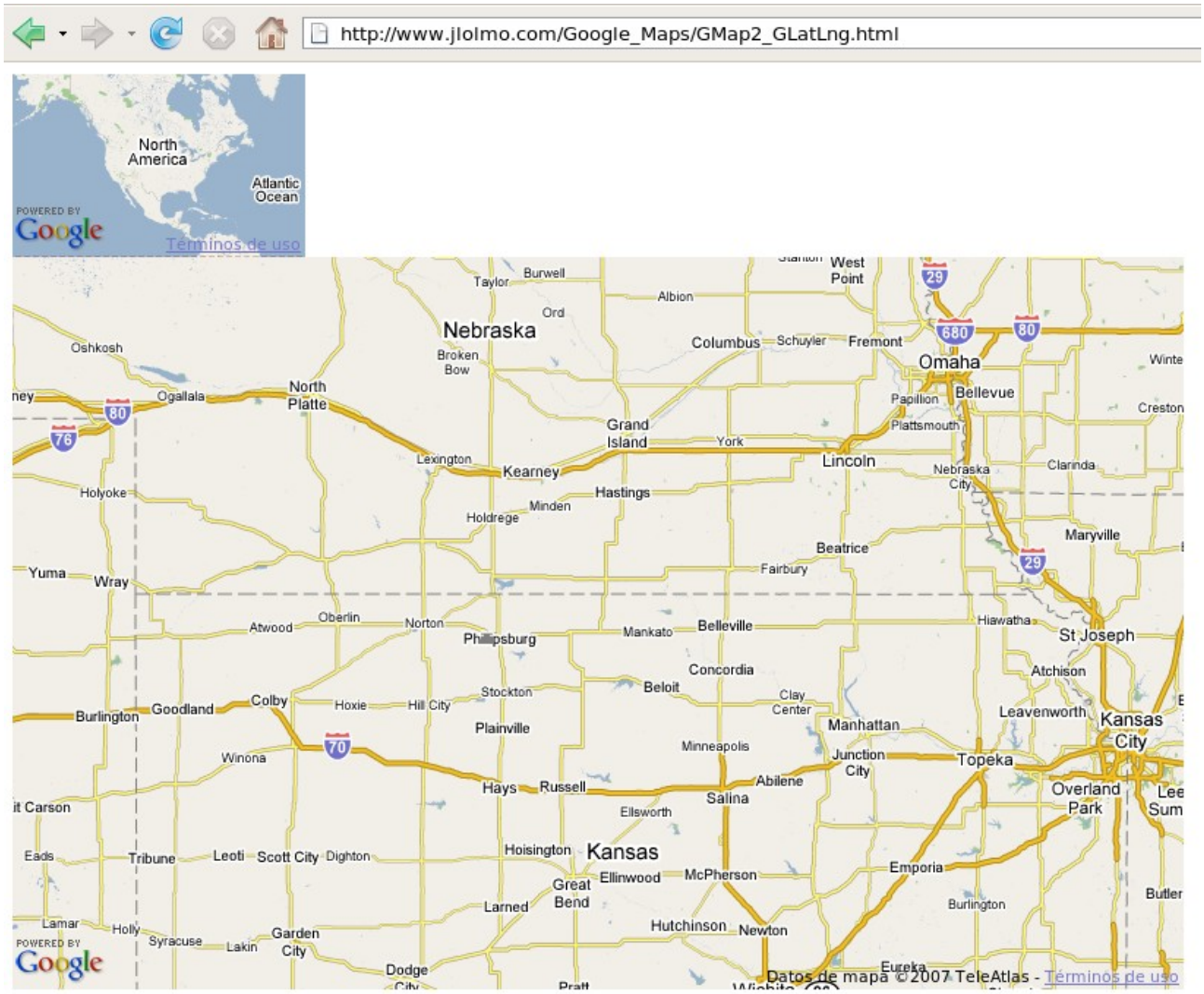

Figura 4.3. Ejemplo de uso de GLatLng y dos objetos GMap2

4.2.3 Controles de mapa

4.2.3.1 Arrastre

El usuario puede arrastrar por defecto cada objeto GMap2 (cada mapa) que situemos en nuestra página. Podemos usar los métodos map.enableDragging()y map.disableDragging() para controlar esta interacción con el usuario. Incluso podemos preguntar por el estado actual utilizando

map.draggingEnabled(), que devolverá true si se puede arrastrar el mapa.

4.2.3.2 Zoom

Para permitir a los usuarios hacer zoom sobre un mapa, hemos de añadir un componente de zoom al mismo. La API proporciona tres opciones para los controles de zoom:

- GLargeMapControl: muestra los 18 niveles de zoom en una barra deslizable y los botones más y menos al principio y al final de la barra.
- GSmallMapControl: ofrece los botones más y menos sin la barra deslizable. También incluye unos botones para el desplazamiento del mapa.
- GSmallZoomControl: únicamente ofrece los botones más y menos.

Para añadir un control de zoom al mapa, utilizaremos map.addControl(new GLargeMapControl()), sustituyéndolo por GSmallMapControl o GSmallZoomControl según proceda.

También es interesante incluir el control de escala, GScaleControl, que proporciona información visual al usuario sobre la escala del mapa.

map.addControl() acepta otro parámetro opcional, GControlPosition, que permite situar el control en el lugar que deseemos, y que recibe dos argumentos:

- Anchor, que será uno de los siguientes valores constantes: G_ANCHOR_BOTTOM_LEFT, G_ANCHOR_BOTTOM_RIGHT, G_ANCHOR_TOP_LEFT o G_ANCHOR_TOP_RIGHT.
- Offset o tamaño, que es un par (x,y) que crea un rectángulo invisible. Se define con GSize(), donde x será la anchura e y la altura.

Ejemplo que añade un control de mapa con la barra deslizable y un control de escala en la esquina inferior derecha de tamaño 20x20.

```
map.addControl(new GLargeMapControl());
map.addControl(new GScaleControl(),
         new GControlPosition(G_ANCHOR_BOTTOM_RIGHT,
                              new GSize(20,20)));
```

4.2.3.3 Cambiar el tipo de mapa

Por defecto se muestra el modo Mapa, que es la vista vectorial. Se puede ajustar el tipo de mapa utilizando map.setMapType(map_type), donde map_type puede ser G_NORMAL_MAP, G_SATELLITE_MAP o G_HYBRID_MAP.
Otra posibilidad es añadir un control para que el usuario pueda cambiar de una vista a otra:

map.addControl(new GmapTypeControl()).

Para determinar el tipo de mapa actual, podemos usar map.getCurrentMapType(), que devolverá un objeto GmapType. Para mostrar el nombre del tipo de mapa actual usamos map.getCurrentMapType().getName().

4.2.3.4 GOverviewMap

Este es otro control que sirve para añadir una vista general del mapa principal. También es interactiva, y si la arrastramos también arrastraremos al mismo tiempo el mapa principal. Para añadirlo, usamos map.addControl(new GOverviewMap()).

4.2.4 *Datos de usuario (objetos personalizados)*

4.2.4.1 GMarker

Un objeto GMarker permite visualizar un punto (marcarlo) en el mapa. El constructor toma un objeto GLatLng como único parámetro requerido.

Una vez tenemos la marca, hemos de decirle al mapa que la muestre: map.addOverlay(myMarker). Todos los objetos que personalizados que situemos sobre el mapa se denominan *overlays*. Podemos eliminar la marca así: map.removeOverlay(myMarker). También podemos eliminar todas las marcas: map.clearOverlays().

```
var myPoint = new GLatLng(38.898748, -77.037684);
var myMarker = new GMarker(myPoint);
map.addOverlay(myMarker);
```

Teóricamente se pueden situar tantas marcas como queramos sobre un mapa, pero el rendimiento decae significativamente a partir de 100 marcas.

4.2.4.2 GIcon

Este objeto es el icono o imagen que usa GMarker para marcar un punto. Es completamente personalizable, y debe ser un fichero PNG (no mayor de un cuadrado de 20-30 píxeles para que no parezca demasiado grande en relación con el resto del mapa).

```
var myIcon = new GIcon();
myIcon.image = "http://www.mapmap.org/googlemaps/google.png";
myIcon.iconSize = new Gsize(16,16);

var myMarker = new GMarker(myPoint, myIcon);
map.addOverlay(myMarker);
```

También se pueden añadir sombras a los iconos con icon.shadow e icon.shadowSize. Si vamos a adjuntar Info Windows a nuestros GIcons, hemos de especificar la propiedad icon.infoWindowAnchor. Por defecto un GIcon adjunta una Info Window arriba en el centro o (10,0).

```
var defaultIcon = new GIcon();
defaultIcon.image = "http://www.google.com/mapfiles/marker.png";
defaultIcon.iconSize = new GSize(20,34);
defaultIcon.shadow = "http://www.google.com/mapfiles/shadow50.png";
defaultIcon.shadowSize = new GSize(37,34);
defaultIcon.iconAnchor = new GPoint(10,34);
defaultIcon.infoWindowAnchor = new GPoint(10,0);
```

La propiedad defaultIcon.iconAnchor representa el pixel exacto del icono que hacemos que coincida con el punto GlatLng.

4.2.4.3 Ventanas informativas

La API las trata de manera diferente a las marcas: mientras que podemos tener tantas marcas como queramos, sólo se podrá desplegar una *Info Window* cada vez. Podemos controlar la capacidad del mapa de mostrar las Info Windows con map.enableInfoWindow() y map.disableInfoWindow(). Para comprobar el estado actual, usamos el método map.infoWindowEnabled().

Una Info Window necesita un punto y un *payload*. Si openInfoWindow() se llama en un GMarker, el punto está implícito. Si no, si se llama en el mapa, hemos de especificar el punto. El *payload* puede ser bien un elemento HTML DOM o bien una string que contenga HTML (map.openInfoWindow() o map.openInfoWindowHtml(), respectivamente).

También tenemos la posibilidad de añadir Info Windows con varias pestañas creando un array de GInfoWindowTabs. Usaremos los métodos openInfoWindowTabs() y openInfoWindowTabsHtml().

```
var geoCenter = map.getCenter(); //returns a GLatLng
var geoBounds = map.getBounds(); //returns a GLatLngBounds
var geoTabText = "Center point: (" + geoCenter.toUrlValue() + ")";
geoTabText += "<br />";
geoTabText += "Bounds: (" + geoBounds.getSouthWest().toUrlValue() + ")";
geoTabText += ", (" + geoBounds.getNorthEast().toUrlValue() + ")";
var pixelCenter = map.fromLatLngToDivPixel(geoCenter); // returns a GPoint
var pixelBounds = map.getSize(); //returns a GSize
var pixelTabText = "Center point: " + pixelCenter.toString();
pixelTabText += "<br />";
pixelTabText += "Bounds: " + pixelBounds.toString();
var tabs = [
  new GInfoWindowTab("geo", geoTabText),
  new GInfoWindowTab("pixel", pixelTabText)
];
map.openInfoWindowTabsHtml(geoCenter, tabs);
```

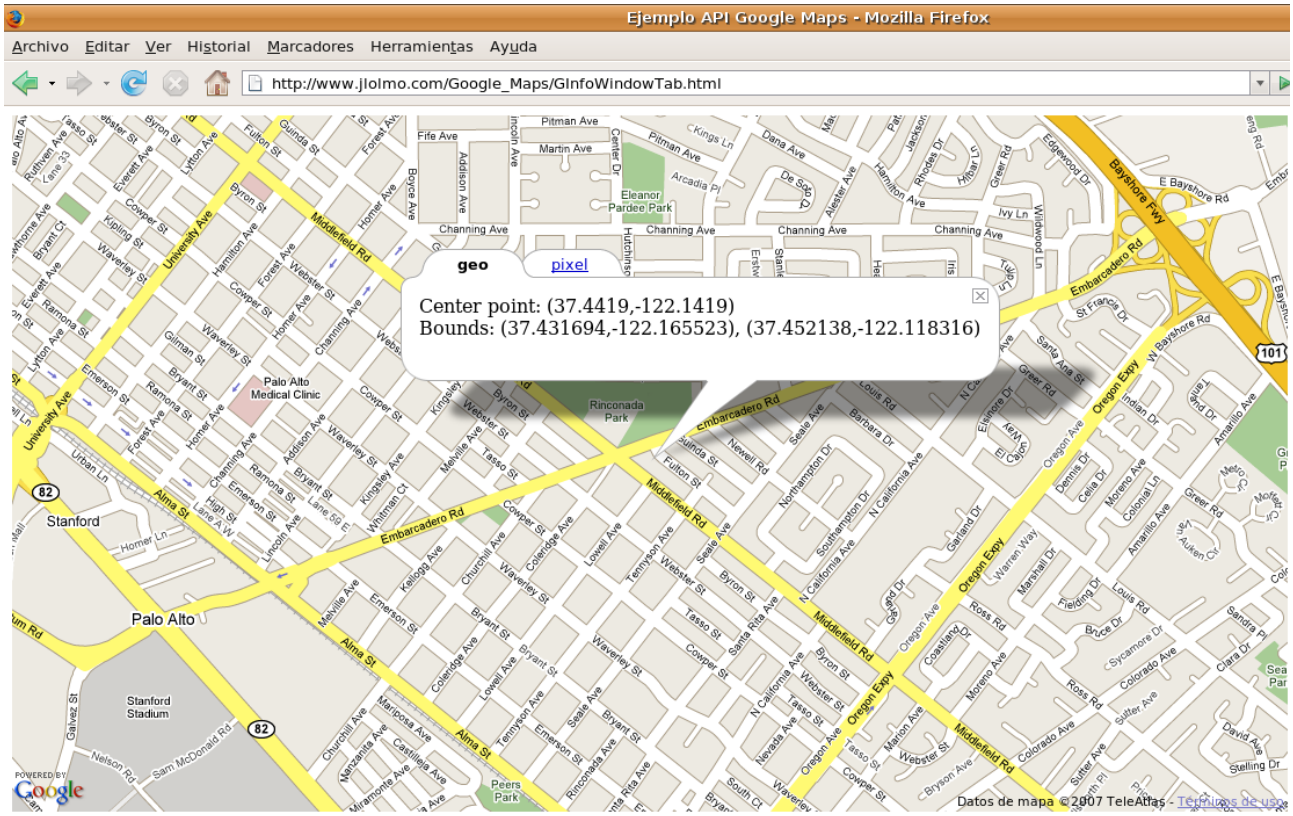

Figura 4.4. Ejemplo de uso de GInfoWindowTab

Además de las ventanas informativas de texto, podemos situar una ventana de información gráfica que muestre una zona del mapa ampliada en un mini mapa. El método showMapBlowup() está disponible para usar sobre el mapa y sobre las marcas. Por defecto, el nivel de zoom que usa es el 17 y el tipo de mapa será el actual. Podemos sobrescribir estos valores pasándole una GinfoWindowOptions.

```
var opts = new Object();
opts.zoomLevel = 14;
opts.mapType = G_SATELLITE_MAP;
googleMarker.showMapBlowup(opts);
//NOTE: rather than creating a separate opts object,
//      you can pass in an anonymous object
googleMarker.showMapBlowup({zoomLevel:14, mapType:G_SATELLITE_MAP});
```

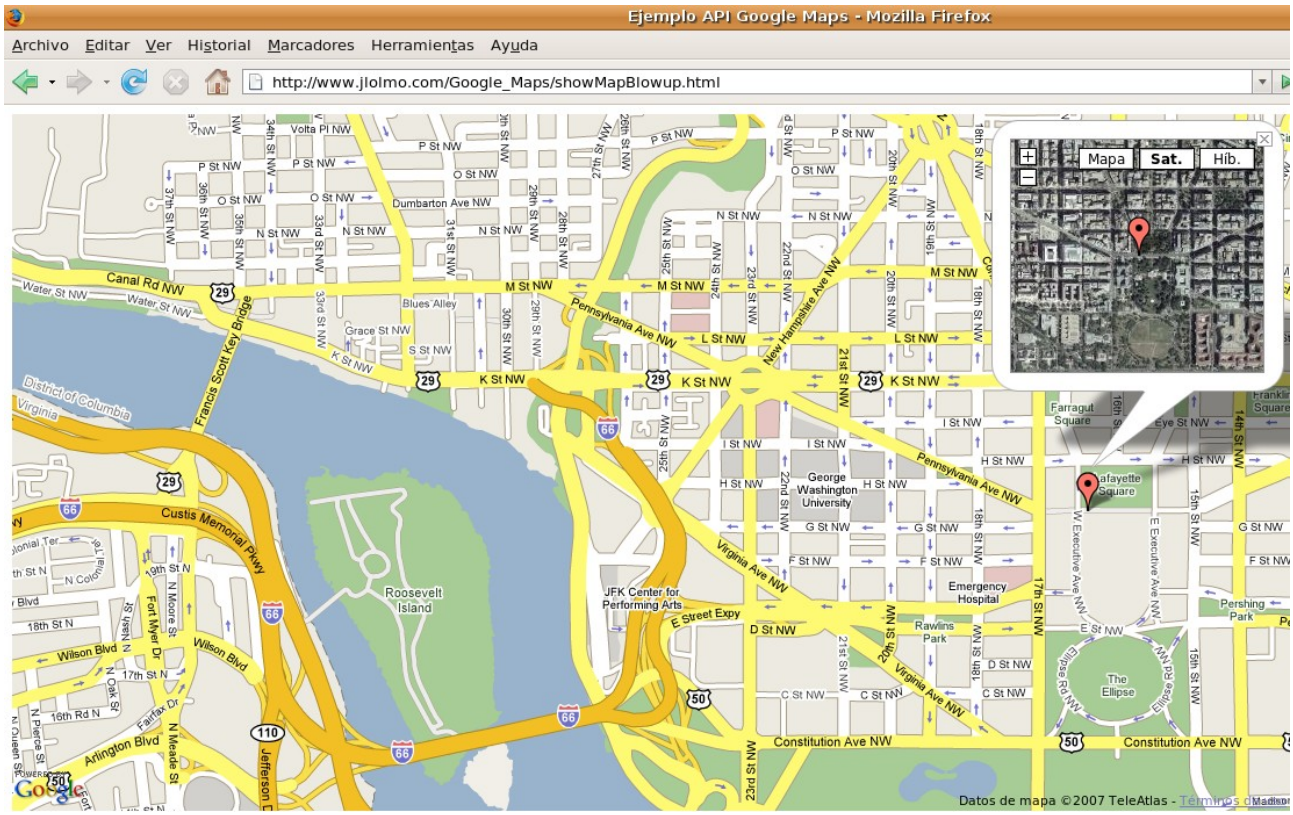

Figura 4.5. Ejemplo de uso de showMapBlowup

4.2.4.4 GPolyline

Para construir una GPolyline hemos de pasarle al constructor un array de GLatLngs. Podemos pasar argumentos opcionales para cambiar la apariencia: color, weight y opacity. El color hemos de expresarlo en estilo HTML hexadecimal (por ejemplo, #ff0000 en vez de rojo). En Internet Explorer, Google Maps utiliza VML para dibujar polilíneas. En los demás navegadores, realizamos una petición de una imagen de la línea al servidor de Google y superponemos (overlay) la imagen en el mapa, refrescando la imagen tanto como sea necesario al hacer zoom o arrastrar el mapa.

El siguiente código crea una polilínea roja de 10 píxeles de anchura entre dos puntos:

```
var polyline = new GPolyline([
    new GLatLng(37.4419, -122.1419),
    new GLatLng(37.4519, -122.1519)
  ], "#ff0000", 10);
map.addOverlay(polyline);
```

4.2.5 Eventos

4.2.5.1 `GEvent`

El modelo de eventos nos permite reaccionar ante alguna acción del usuario. La API de Google Maps usa los métodos modernos para añadir "escuchadores de eventos" (event listeners) a los elementos.

```
var clickListener = GEvent.addListener(map, "click", doSomething());
GEvent.removeListener(clickListener);
```

4.2.5.2 `GBrowserIsCompatible`

La API nos ofrece un método para comprobar si el navegador es compatible:

```
if (GBrowserIsCompatible()) {
  var map = new GMap2(document.getElementById("map"));
  map.setCenter(myPoint, 1);
}
```

4.2.5.3 Eventos `GMap`

La mayoría de los eventos en los objetos maps son simplemente eventos que no devuelven ningún valor, sino que tan sólo lanzan una notificación:

- `drag()` : se está enviando continuamente cuando el usuario arrastra el mapa
- `dragstart()` : se envía cuando el usuario empieza a arrastrar el mapa
- `dragend()` : se envía cuando el deja de arrastrar el mapa usuario
- `infowindowopen()` : se envía cuando se muestra una *Info Window*
- `infowindowclose()` : se envía cuando se cierra una *Info Window*
- `maptypechanged()` : se envía cuando el usuario cambia el tipo de mapa

Otros eventos sí que devuelven información adicional:

- `click(overlay, point)` : se envía cuando se hace click sobre el mapa. Si el usuario hace click sobre una GMarker u otro objeto *overlay*, se retorna el objeto. Si hace click sobre un área abierta del mapa, se devuelve el punto lat/long del click.

- `zoomend(previousZoomLevel, currentZoomLevel)` : se envía cuando el usuario cambia el nivel de zoom del mapa. Devuelve el nivel de zoom prevío y el actual.
- `addoverlay(overlay)` : se envía cuando se añade un *overlay* al mapa. Se devuelve el objeto overlay añadido.
- `removeoverlay(overlay)` : se envía cuando se elimina un *overlay* del mapa. Se devuelve el objeto *overlay* eliminado.
- `clearoverlays()` : se envía cuando se eliminan todos los *overlays* del mapa
- `mousemove(latlng)` : se envía continuamente cuando el ratón está en movimiento. Se devuelve el punto lat/long del cursor.
- `mouseout(latlng)` : se envía cuando el ratón se sale del mapa. Se devuelve el punto lat/lng del cursor.
- `mouseover(latlng)` : se envía cuando el ratón entra en el mapa. Se devuelve el punto lat/lng del cursor.

4.2.5.4 Manejadores de eventos

Los eventos se pueden pasar a funciones o pueden ser manejados por funciones en línea (si el código del manejador del evento no es más de un par de líneas y no se va a reutilizar) :

```
var dragListener = GEvent.addListener(map, "drag", function() {
    document.getElementById("output").innerHTML =
                    map.getCenter().toUrlValue();
    } );
var clickListener = GEvent.addListener(map, "click",
  function(overlay, point) {
    handleMapClick(overlay, point);
    } );
```

4.2.5.5 Eventos `GMarker`

Dependiendo de donde registremos el event listener, se retornarán unos argumentos u otros. Por ejemplo, si registrarmos un click listener sobre el mapa, el listener devolverá el *overlay* o el punto en el evento (dependiendo de si hacemos click sobre un overlay o un área vacía); si registramos un click listener en un único punto, el evento no tiene argumentos asociados porque sabemos exactamente qué punto se ha clickado.

- `click()` : se envía cuando se hace click sobre un GMarker
- `dblclick()` : se envía cuando se hace doble-click sobre un Gmarker
- `dragstart()` : se envía cuando se comienza a arrastrar un GMarker (si tiene esta

capacidad)

- `drag()` : se envía mientras se está arrastrando un GMarker
- `dragend()` : se envía cuando termina de arrastrarse un GMarker
- `mouseover()` : se envía cuando el ratón pasa sobre un GMarker
- `mouseout()` : se envía cuando el ratón sale de un GMarker
- `infowindowopen()` : se envía cuando se muestra la *Info Window* correspondiente
- `infowindowclose()` : se envía cuandos se cierra la *Info Window* asociada
- `remove()` : se envía cuando se elimina un GMarker del mapa

Ejemplo: situamos una marca arrastrable en el mapa, y escuchamos un par de eventos. Por defecto, las GMarker se pueden clickar pero no arrastrar, por lo que hemos de inicializarla con la opción draggable a true.

```
var map = new GMap2(document.getElementById("map"));
var center = new GLatLng(37.4419, -122.1419);
map.setCenter(center, 13);
var marker = new GMarker(center, {draggable: true});

GEvent.addListener(marker, "dragstart", function() {
  map.closeInfoWindow();
});

GEvent.addListener(marker, "dragend", function() {
  marker.openInfoWindowHtml("Just bouncing along...");
});

map.addOverlay(marker);
```

4.2.6 AJAX

4.2.6.1 `GXmlHttp`

Aunque el objeto XMLHttpRequest está presente en la mayoría de los navegadores, su implementación suele ser incompatible entre unos y otros. Por tanto, es tarea del desarrollador normalizar el modelo AJAX entre los navegadores:

```
//1. Mozilla, Safari, et al
//2. IE
var req;
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
}
else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
}
```

La API nos proporciona un objeto, GxmlHttp, que normaliza el modelo de eventos de AJAX existente entre los navegadores por nosotros:

```
var req = GXmlHttp.create();
```

4.2.6.2 Servicios web de geocodificación y objeto `GClientGeocoder`

Podemos usar el objeto GXmlHttp para realizar cualquier llamada AJAX. Por ejemplo, puede sernos útil utilizarlo para buscar las coordenadas de una ciudad. Así, tendremos la lista en un servidor y descargaremos los datos que necesitemos a través de una petición al servicio web.

Los servicios web que aceptan consultas que devuelven las coordenadas polares (lat/long) de una dirección dada se denominan *geocoders* (geocodificadores). Podemos usar el objeto GXmlHttp para realizar llamadas al geocoder que escojamos.

No obstante, la API de Google Maps también nos ofrece su geocoder, que podemos acceder bien vía HTTP o desde dentro del código JavaScript:

Para acceder al geocoder desde JavaScript, usamos el objeto GClientGeocoder. Utilizamos el método getLatLng para convertir una dirección en un objeto GLatLng. En el ejemplo siguiente, hacemos la traducción de una dirección, añadimos una marca en el punto y abrimos una Info Window mostrando la dirección:

```
var map = new GMap2(document.getElementById("map"));
var geocoder = new GclientGeocoder();

function showAddress(address) {
  geocoder.getLatLng(
    address,
    function(point) {
      if (!point) {
        alert(address + " not found");
      } else {
        map.setCenter(point, 13);
        var marker = new GMarker(point);
        map.addOverlay(marker);
        marker.openInfoWindowHtml(address);
      }
    }
  );
}
```

4.2.7 Modificando la API: mostrar capas WMS sobre Google Maps

Existen otras herramientas curiosas desarrolladas por los usuarios que permiten extender la funcionalidad de Google Maps. De hecho, una en concreto nos permite cargar capas de cualquier WMS sobre la API de Google Maps *[04_04]*. Se trata de una librería JavaScript llamada gmap, escrita por Just van den Broecke, que implementa una función que nos permite pasar la url, las capas, etc. de forma sencilla, aprovechándose de una función que genera el botón de cambiar el tipo de mapa.

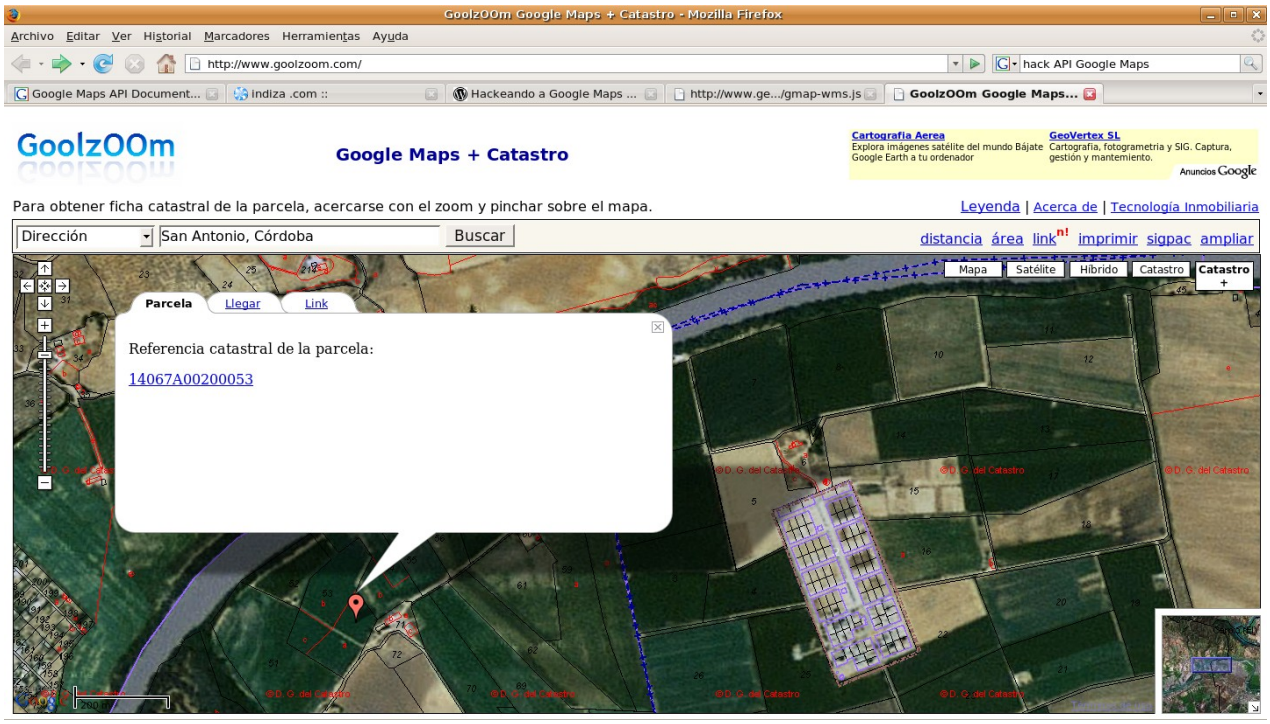

Figura 4.6. Superposición de capas WMS sobre Google Maps

Utilizando la librería Gmap se han creado aplicaciones o *mashups* de Google Maps como *GoolzOOm* (http://www.goolzoom.com), en el que se aúna la tecnología de Google Maps con la capacidad de superponer capas WMS para crear una aplicación que facilita el acceso al catastro español. Esta aplicación interactúa con los servidores públicos del Ministerio del Catastro, con lo que las fichas y los mapas catastrales están continuamente actualizados.

4.3 Aplicaciones de Google Maps

En el apartado anterior hemos comentado una aplicación de Google Maps, GoolZoom, y hemos dicho que se trataba de un *mashup*. Pero, ¿qué es exactamente un *mashup*?

Un *mashup*, en el contexto web 2.0, es un sitio web que accede a datos o servicios de terceros y los combina para crear una nueva aplicación *[04_05]*. Para considerarse *mashup*, el sitio web debe

mezclar datos y servicios de como mínimo dos fuentes distintas, una de las cuales debe ser externa al *mashup*. Un ejemplo será, por tanto, un sitio web que muestre un mapa de Google con información geolocalizada del propio sitio.

Además del *mashup* mencionado en el apartado anterior, podemos encontrar muchas otras modificaciones de Google Maps en la red.

Por ejemplo, buscar casa directamente en un mapa es mucho más sencillo en *Geoviviendas* (http://www.geoviviendas.com), que ha adaptado Google Maps para sus propios intereses, e incluso permite búsquedas dentro de los mapas.

Otra aplicación curiosa, en este caso norteamericana, es *Aviation Wheather Maps*, que muestra los aeropuertos con las condiciones meteorológicas actuales. La podemos encontrar en http://maps.avnwx.com.

WhoIsSick (http://www.whoissick.com) es un nuevo *mashup* de Google enfocado en mostrar información de la salud a los usuarios, rastreando y monitorizando las enfermedades actuales de la gente. Esta aplicación permite filtrar qué enfermedades afectan actualmente a la gente de un determinado barrio, por ejemplo:

Figura 4.7. Mashup WhoIsSick

GeoIPTool (http://www.geoiptool.com) permite conocer información acerca de la IP introducida, tal como: nombre del host, país, ciudad, longitud, latitud...

Otro mashup basado en Google Maps que cabe mencionar es *Wikiloc* (http://www.wikiloc.com) que es un *mashup* de mapas gratuito para compartir rutas con GPS en la web. Utilizando software libre y la API de Google Maps, Wikiloc hace la función de base de datos personal de localizaciones GPS. Desde cualquier punto de acceso a internet un usuario de GPS puede cargar sus datos GPS y al momento visualizar la ruta y waypoints con distinta cartografía de fondo, incluidos servidores de mapas externos WMS. Paralelamente se muestra el perfil de altura, distancia, desniveles acumulados y las fotos o comentarios que el usuario quiera añadir.

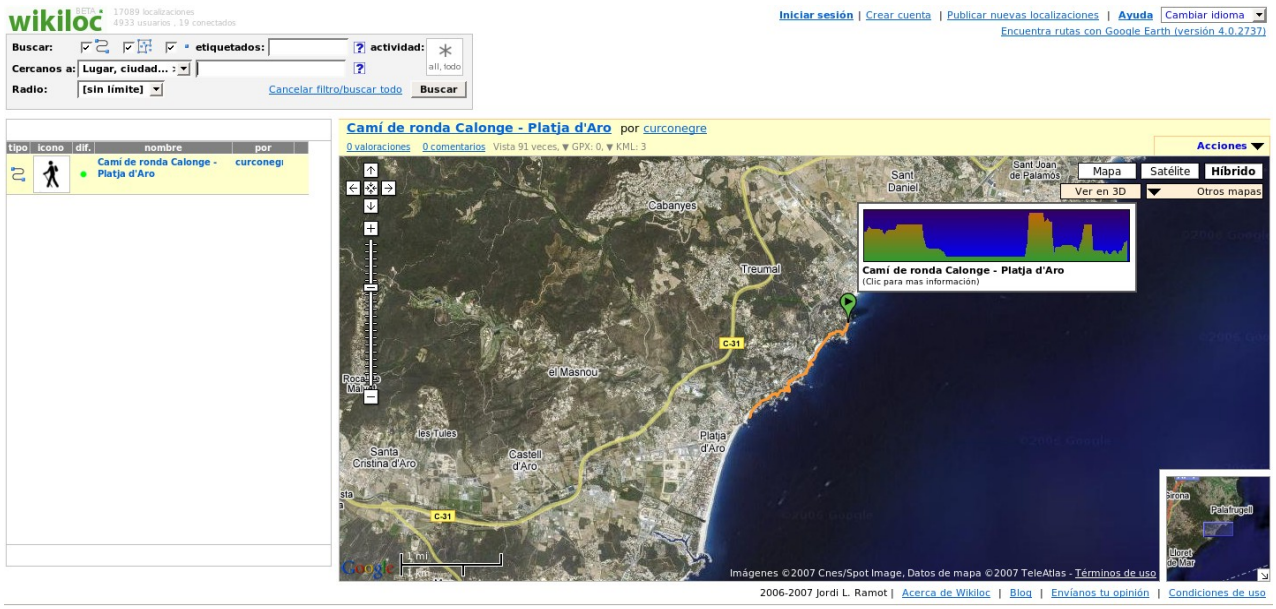

Figura 4.8. Mashup Wikiloc

5 Diseño de la aplicación web

5.1 Introducción

En este apartado se estudiará el sistema software haciendo hincapié en el modelado de datos o descripción de la información, en donde se analizará el sistema desde un punto de vista estático mediante el análisis de la información que es necesario mantener. Para describir estos objetos de datos se utilizará el modelado UML.

Partiendo de la especificación anterior, a continuación se definirá lógica y físicamente el dominio de la información, exponiendo los pasos seguidos para transformar los objetos presentes en el modelo UML a objetos válidos en el modelo lógico (modelo relacional).

A continuación se mostrará la secuencia de activación de ventanas de la aplicación y la arquitectura de la misma, indicando mediante esquemas el proceso de solicitud de información al servidor en modo asíncrono mediante la tecnología AJAX.

Por último, se describirá la interfaz del sistema, enumerando las características de que dotaremos a la misma.

5.2 Modelo conceptual

Se introducen las claves artificiales que se consideran oportunas para solventar las posibles ausencias de identificadores del modelo creado.

La nomenclatura que se sigue para el nombre de las clases es anteponer una 't' de *table* (tabla) delante del nombre de la clase, que estará en inglés. Para los atributos se sigue la misma filosofía, pero dependiendo de lo que preceda al término anglosajón del atributo, el tipo será uno u otro: 'id' indica *identifier* (identificador), 'tx' *text* (texto o cadena), 'i' *integer* (entero), 'f' *float* (flotante), 'b' *boolean* (booleano), y 'fk' *foreign key* (clave foránea).

Clase tCustomer: representa a un cliente de la empresa de seguridad que contrata sus servicios. Características:

- Nombre de la clase: tCustomer
- Atributo identificador principal: txCompanyName

- Atributo identificador alternativo: txCIF
- Número de atributos: 6

Clase tCamera: representa a una cámara de un cliente (customer). tCamera es débil por identificación con respecto a tCustomer, y además, al no disponer de identificador unívoco, hemos de añadir un identificador artificial. Cada cliente podrá tener n cámaras. De cada cámara se almacena el nombre:

- Nombre de la clase: tCamera
- Atributo identificador principal: idCamera
- Atributo identificador alternativo: ninguno
- Número de atributos: 2

Clase tAgent: representa a un agente móvil. Consideramos que un agente de seguridad tiene asignada una única ruta.

- Nombre de la clase: tAgent
- Atributo identificador principal: txNIF
- Atributo identificador alternativo: ninguno
- Número de atributos: 5

Clase tRoute: representa a una posible ruta que siguen los agentes móviles. Está identificada por la clave artificial idRoute, y tiene un nombre único. Consideramos que una determinada ruta sólo se le asigna a un único agente de seguridad.

- Nombre de la clase: tRoute
- Atributo identificador principal: idRoute
- Atributo identificador alternativo: ninguno
- Número de atributos: 2

Clase tRoutePoint: se trata de un tipo de entidad débil por identificación con respecto a la entidad tRoute, y representa a un punto de una ruta. Una ruta estará compuesta por 2 o más puntos, identificados por las coordenadas geográficas (latitud y longitud). Además, nos hace falta conocer el orden en que el agente pasa por dicho punto de una ruta, para lo cuál almacenamos el orden del punto en dicho recorrido. Por último, necesitamos saber si este punto de la ruta es el punto donde se encuentra actualmente el agente, por lo que usamos un atributo booleano para este propósito.

- Nombre de la clase: tRoutePoint
- Atributo identificador principal: fLat + fLong + idRoute (al ser un caso de dependencia en identificación la clave primaria de la relación en la que se ha transformado la entidad débil debe estar formada por la concatenación de las claves de las dos entidades participantes.

- Atributo identificador alternativo: ninguno
- Número de atributos: 5

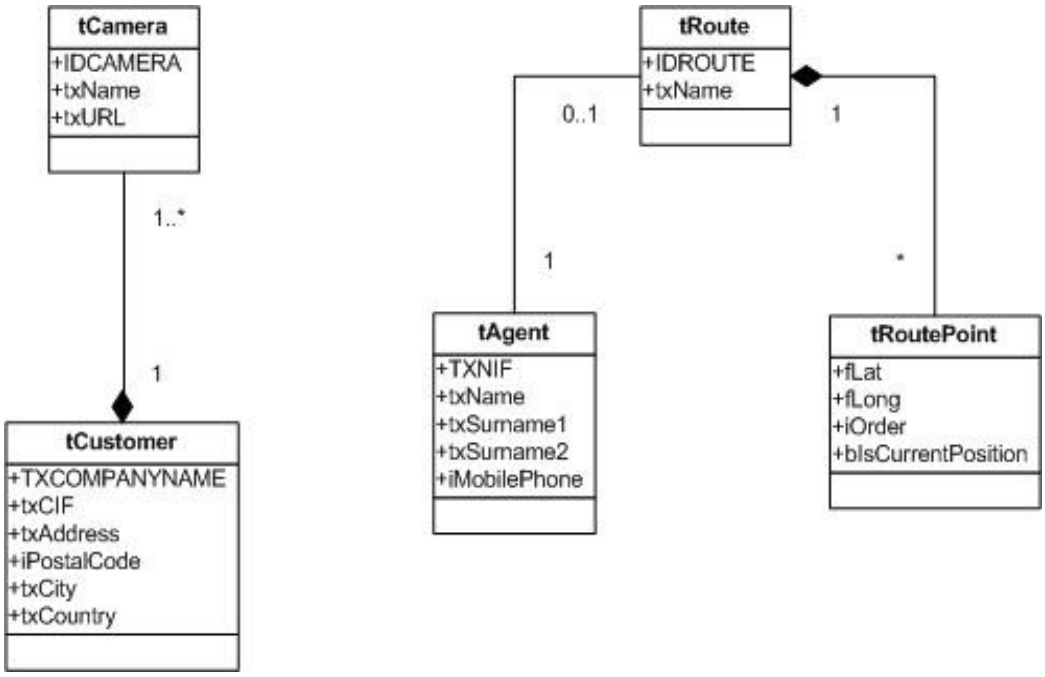

Figura 5.1. Diagrama de clases UML

5.3 Modelo lógico

Para facilitar el acceso a la información y las consultas, finalmente se decide introducir claves artificiales para el resto de relaciones, a pesar de que las claves originales no son pesadas (no están formadas por un gran número de atributos). A continuación enumeramos las diferentes relaciones:

Tabla tCustomer: el tipo de entidad tCustomer se transforma en una relación que mantiene el mismo número de atributos. Se le añade el identificador artificial idCustomer, y se añaden las restricciones UNIQUE y NOT NULL a los atributos identificadores alternativos txCompanyName y txCIF. Tampoco se permiten valores nulos en el resto de atributos.

tCustomer (***idCustomer***, ***txCompanyName***, ***txCIF***, ***txAddress***, ***iPostalCode***, ***txCity***, ***txCountry***)

Tabla tCamera: el tipo de entidad tCamera se transforma en una relación que lleva el mismo nombre, y de este tipo de entidad toma los atributos idCamera, txName y txURL. Además, al mantener una

interrelación de tipo 1:N con respecto a tCustomer, se propaga la clave idCustomer desde el tipo de entidad tCustomer a tCamera, pero con el nombre fkCustomer para seguir la nomenclatura establecida. fkCustomer no podrá tomar valores nulos, ya que la cardinalidad mínima de tCamera con tCustomer es 1. El resto de atributos tampoco pueden ser nulos.

tCamera (***idCamera**, txName, txURL, fkCustomer*)

Tabla tRoute: el tipo de entidad tRoute se transforma en una relación que lleva el mismo nombre e igual número de atributos.

tRoute (***idRoute**, txName*)

Tabla tAgent: el tipo de entidad tAgent se transforma en una relación que lleva el mismo nombre. La relación toma todos los atributos de la entidad tAgent, excepto idAgent que es una clave artificial que añadimos al transformar al modelo relacional, y fkRoute que se propaga desde el tipo de entidad tRoute, con el cual mantiene una interrelación 1:1 (Nota: al tratarse de un tipo de interrelación 1:1, podríamos haber optado por propagar la clave en el sentido contrario, es decir, propagar idAgent a tRoute, o bien por haber creado una tabla intermedia). El único atributo donde se permiten valores nulos es txSurname2.

tAgent (***idAgent**, **txNIF**, txName, txSurname1,* *txSurname2*, *iMobilePhone, fkRoute*)

Tabla tRoutePoint: el tipo de entidad tRoutePoint se transforma en una relación que lleva el mismo nombre e igual número de atributos. Ya comentamos en el modelo conceptual que mantiene una debilidad por identificación con respecto a tRoute, y, dado que no añadimos ninguna clave artificial para identificar unívocamente a una tupla de esta relación, la clave que se propaga desde tRoute formará parte de la clave primaria de esta tabla. Ningún atributo podrá ser nulo.

tRoutePoint (***fkRoute, fLat, fLong**, iOrder, bIsCurrentPosition*)

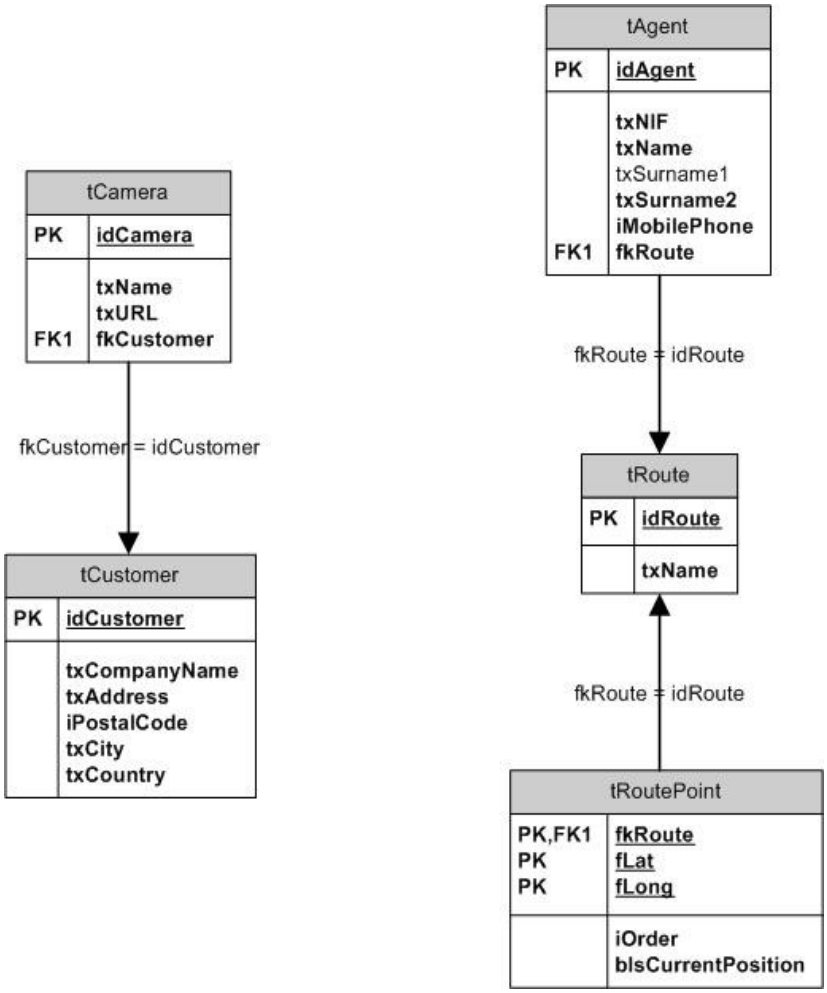

Figura 5.2. Esquema relacional

5.4 Script .sql del SGBD MySQL

En este apartado se adjunta el script .sql de creación de la base de datos en el SGBD MySQL. Este fichero contendrá las tablas resultantes del proceso de transformación al modelo relacional a que sometimos en el apartado 3.3 a los distintos tipos de entidades que representan los datos de nuestro sistema; es decir, contendrá la intención de la base de datos.

```
CREATE DATABASE PFC2;

USE PFC2;

--
```

```
-- Estructura de tabla para la tabla 'tCustomer'
--

CREATE TABLE IF NOT EXISTS tCustomer (
      idCustomer INT AUTO_INCREMENT,
      txCompanyName VARCHAR(50) NOT NULL,
      txCIF VARCHAR(9) NOT NULL,
      txAddress VARCHAR(50) NOT NULL,
      iPostalCode INT(5) NOT NULL,
      txCity VARCHAR(15) NOT NULL,
      txCountry VARCHAR(15) NOT NULL,
      CONSTRAINT PK_tCustomer PRIMARY KEY (idCustomer),
      CONSTRAINT UK1_tCustomer UNIQUE KEY (txCompanyName),
      CONSTRAINT UK2_tCustomer UNIQUE KEY (txCIF)
);



--
-- Estructura de tabla para la tabla 'tCamera'
--

CREATE TABLE IF NOT EXISTS tCamera (
      idCamera INT AUTO_INCREMENT,
      txName VARCHAR(25) NOT NULL,
      txURL VARCHAR(100) NOT NULL,
      fkCustomer INT NOT NULL,
      CONSTRAINT PK_tCamera PRIMARY KEY (idCamera),
      CONSTRAINT FK1_tCamera FOREIGN KEY (fkCustomer)
        REFERENCES tCustomer (idCustomer),
      CONSTRAINT UK1_tCamera UNIQUE KEY (txName,fkCustomer)
);

--
-- Estructura de tabla para la tabla 'tRoute'
--

CREATE TABLE IF NOT EXISTS tRoute (
      idRoute INT AUTO_INCREMENT,
      txName VARCHAR(30) NOT NULL,
      CONSTRAINT PK_tRoute PRIMARY KEY (idRoute)
);


--
-- Estructura de tabla para la tabla 'tAgent'
--

CREATE TABLE IF NOT EXISTS tAgent (
      idAgent INT AUTO_INCREMENT,
      txNIF VARCHAR(9) NOT NULL,
      txName VARCHAR(20) NOT NULL,
```

```
	txSurname1 VARCHAR(20) NOT NULL,
	txSurname2 VARCHAR(20) DEFAULT NULL,
	iMobilePhone INT(9) NOT NULL,
	fkRoute INT NOT NULL,
	CONSTRAINT PK_tAgent PRIMARY KEY (idAgent),
	CONSTRAINT FK1_tAgent FOREIGN KEY (fkRoute)
	  REFERENCES tRoute (idRoute),
	CONSTRAINT UK1_tAgente UNIQUE KEY (txNIF),
	CONSTRAINT UK2_tAgente UNIQUE KEY (iMobilePhone)
);


--
-- Estructura de tabla para la tabla 'tRoutePoint'
--

CREATE TABLE tRoutePoint (
	fkRoute INT NOT NULL,
	fLat FLOAT NOT NULL,
	fLong FLOAT NOT NULL,
	iOrder INT NOT NULL,
	bIsCurrentPosition TINYINT(1) NOT NULL,
	CONSTRAINT PK_tRoutePoint PRIMARY KEY (fkRoute,fkLat,fkLong),
	CONSTRAINT FK1_tRoutePoint  FOREIGN KEY (fkRoute) REFERENCES tRoute
(idRoute)
);
```

5.5 Diseño arquitectónico

El diseño arquitectónico para los sistemas y aplicaciones basados en web se centra en la definición de la estructura global hipermedia para la WebApp *[05_01]*.

En nuestra aplicación únicamente consideramos una ventana principal donde se muestra el listado de clientes y el listado de agentes. El usuario puede pulsar sobre cualquier fila del listado para que se cargue en el visor de Google Maps la información asociada a la misma. Esta información la recupera el servidor en modo asíncrono de la base de datos de la aplicación.

Por tanto, no tiene mucho sentido mostrar diagramas de activación de ventanas, puesto que sólo existe una ventana principal, pero sí que podemos hablar de interacción con el servidor y cómo se muestran los datos que se recuperan en modo asíncrono en el visor de Google Maps, sin recargar la página, por medio de DHTML. Esto es posible gracias a la etiqueta HTML <DIV>, con la que dividimos nuestra página principal y creamos diferentes capas, cada una con distinto comportamiento. Una de ellas será el visor de Google Maps.

A continuación podemos ver un esquema sencillo que explica el proceso:

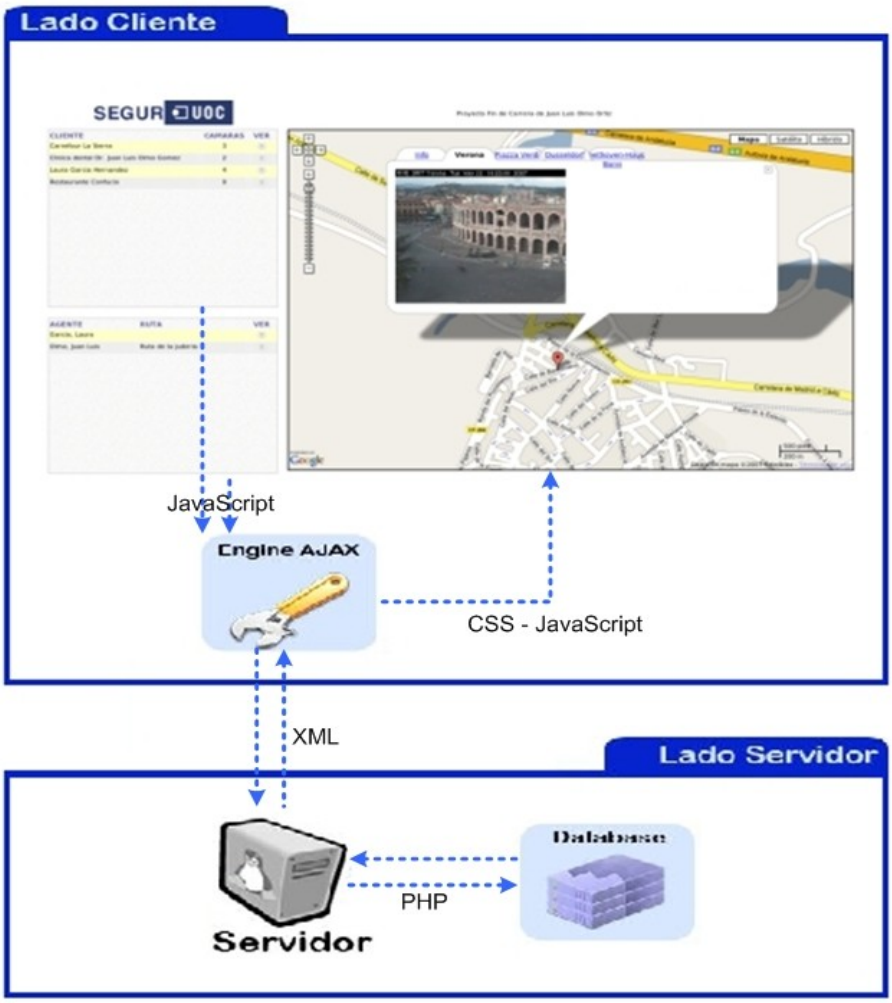

Figura 5.3. Arquitectura del sistema

Podemos observar cómo el usuario, al querer ver la información de algún elemento de los listados, pasa la información mediante JavaScript al motor de AJAX. A continuación, éste se comunica con el servidor, donde se ejecuta uno de los dos scripts PHP que recuperan la información de la base de datos y la devuelven en formato XML al motor de AJAX. Finalmente, se visualiza la información en el visor de Google Maps, gracias a CSS y la API de Google Maps (JavaScript). Todo este proceso se produce de forma asíncrona y sin recargar la página, de modo que para el usuario simula el comportamiento de una aplicación de escritorio.

5.6 Diseño de la interfaz

En este apartado se enumeran las características de que dotaremos a la interfaz de la aplicación. Para

que el usuario final pueda interactuar con nuestro sistema software, éste debe proporcionar los mecanismos para que el usuario pueda realizar diferentes acciones. En nuestro caso, dado que el objetivo del PFC no es otro que trabajar con la API de Google Maps y la tecnología AJAX, se ha decidido no dotar de funcionalidad a la aplicación para que el usuario pueda insertar información, modificarla o borrarla, sino que dicha información ya se encuentra cargada de antemano en la base de datos y el usuario únicamente puede consultarla.

Es fundamental que la interfaz a diseñar sea una interfaz entendible y fácil de usar, en la que los elementos se encuentren dispuestos de forma simple y ordenada. Por ello, se creará una interfaz gráfica en la que trataremos de facilitar al usuario la utilización del sistema, tratando pues de:

- Dejar el control en manos del usuario
- Reducir la carga de memoria del usuario
- Construir una interfaz consecuente

5.6.1 Ergonomía de la interfaz

La interfaz que pretendemos desarrollar deberá cumplir una serie de características en función de la solución por la que nos hemos decantado para resolver el problema y de la naturaleza del problema en sí. Además, una interfaz bien diseñada mejora la percepción del contenido o de los servicios de usuario que proporciona la aplicación web. Así pues, se propondrán a continuación las características que se crean convenientes para dotar al sitio web de una interfaz bien estructurada y ergonómica.

5.6.1.1 Ubicación de la aplicación en la pantalla

Por el uso que se le puede dar a la aplicación, la ubicación de la ventana ocupará toda la pantalla. Se diseñará de modo que sea compatible con cualquier resolución, sin que se pierda información en ninguna de ellas. No obstante, el usuario de la aplicación web podrá redimensionar la pantalla o bien minimizarla, como bien se puede hacer desde cualquier explorador web, que es al fin y al cabo donde se va a ejecutar la aplicación.

5.6.1.2 Información a mostrar, formato y situación de la misma en la pantalla

El objetivo de la interfaz es que el usuario pueda interactuar con ella de forma sencilla, eficiente y amigable, reduciendo la carga de memoria. Para ello, se mostrará la información de la siguiente forma:

- El estilo y el formato de la interfaz será consistente y atractivo, presentando la información de forma clara y ordenada.
- Los iconos se colocarán de forma intuitiva para el usuario, el título de la ventana será representativo de la información que trata, así como las capas donde se listarán los clientes y los agentes móviles, que estarán bien diferenciadas.
- La información que se visualizará en pantalla será muy significativa y relevante. No se mostrarán datos sin sentido, minimizándose así la cantidad de información que el usuario debe memorizar.
- La información se mostrará al usuario de forma clara, con resaltado de las palabras importantes y con un correcto uso de las tabulaciones, las mayúsculas y las minúsculas.
- El usuario podrá acceder a toda la funcionalidad de la aplicación desde una única ventana. En ella, en la parte izquierda y a modo de menú se dividirá la pantalla en dos capas: en una se mostrará el listado de clientes y en otra el de agentes móviles. El resto de la ventana estará ocupada por el visor de Google Maps, donde se presentarán al usuario las cámaras de los clientes o los agentes móviles cargados de la base de datos, según seleccione un elemento del listado de las capas antes mencionadas.
- Se evitará al usuario tener que recorrer la pantalla. La información se incluirá dentro de las proporciones y dimensiones normales de una ventana del navegador. No obstante, al estar diseñada cada capa mediante hojas de estilo en cascada, las dimensionaremos mediante porcentajes y no valores estáticos, por lo que en función de que el usuario modifique el tamaño de la ventana del navegador, el tamaño de estas capas se ajustará a la nueva dimensión del mismo, sin dejar de mostrar la información de forma adecuada.

5.6.1.3 Utilidad y función del color en la aplicación

Al igual que dotaremos de interactividad a la interfaz por medio del lenguaje JavaScript y, especialmente, a través de AJAX, simulando que se trata de una aplicación de escritorio, para hacer aún más atractiva e intuitiva la interfaz usaremos hojas de estilo en cascada (*Cascade Style Sheets*, CSS). Así, mediante programación se definirán una serie de clases que luego se podrán aplicar a la información según sea conveniente, determinando así su color, tamaño, tipo de letra, estilo... . Especialmente significativa es la función del color en el resaltado de información relevante, de manera que el usuario pueda localizarla rápidamente y sin apenas esfuerzo ni carga de memoria.

Se ha decidido usar como color de fondo en las capas donde se listan los clientes y los agentes móviles un tono grisáceo, para dotar así de mayor neutralidad a la aplicación, de modo que el texto resalte del fondo adecuadamente. Además, dicho color es más familiar y bastante común en muchas aplicaciones.

Por otro lado, en los listados de clientes y agentes móviles se diferenciarán muy bien los elementos, ya que alternativamente se modificará el color de fondo de modo que haya una entrada con color gris y

otra con amarillo.

También hay que destacar que la información se encuentra bien separada en tres capas distintas: una será el listado de clientes, otra el listado de agentes móviles, y la otra el visor de Google Maps.

5.6.2 Descripción de la interfaz

En este apartado mostraremos de manera esquemática cómo se distribuye la interfaz de usuario y los principales elementos que la integran.

- Logo y cabecera: capa CSS donde incluimos el logotipo de la aplicación (*SegurUOC*) y, a su derecha, la leyenda "Proyecto Fin de Carrera de Juan Luis Olmo Ortiz".
- Listado de clientes: se trata de una capa CSS que se coloca en la parte superior izquierda de la ventana. En él se presenta el listado de clientes de la empresa de seguridad, cada uno con el número de cámaras que tiene contratadas y un botón con forma de ojo que permite "ver" y localizar la información del cliente y sus cámaras en el visor de Google Maps.
- Listado de agentes móviles: al igual que en el caso de los clientes, se trata de otra capa donde mostramos el nombre de cada agente y el nombre de la ruta o recorrido que vigila. También acompaña a cada elemento del listado un botón ver, para mostrar su ruta en el visor de Google Maps y la situación actual del agente. Esta capa tendrá menor altura que la de listado de clientes, pues suponemos que habrá un número más o menos fijo de agentes móviles, o su número crecerá más lentamente que el de clientes de la empresa.
- Visor de Google Maps: es el 'contenido' de la aplicación. En él se localizarán en el mapa la situación de las cámaras de los clientes y se podrán visualizar las mismas, o bien se mostrará la posición actual de los agentes de seguridad y el recorrido que realizan.

En la figura 6.4 podemos ver un esquema con los elementos que acabamos de describir.

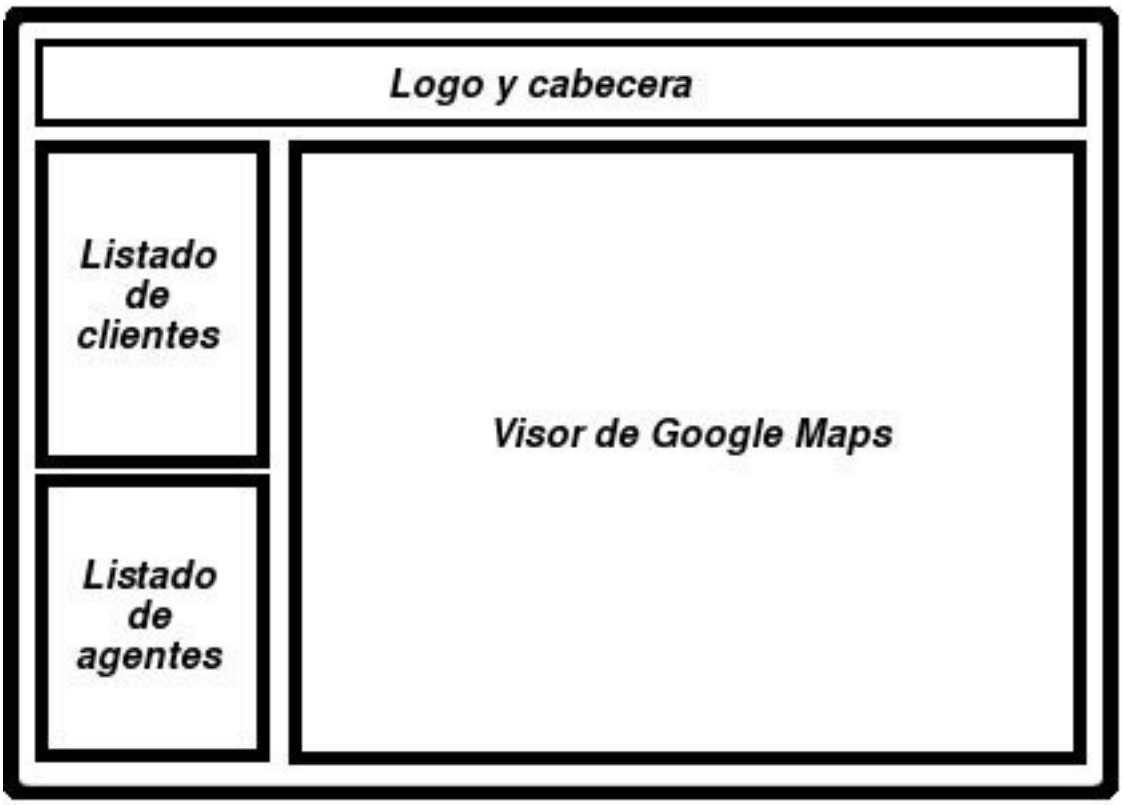

Figura 5.4. Descripción esquemática de la interfaz

A continuación se muestran los objetos de que se compondrá la interfaz para permitir y facilitar al usuario realizar las distintas acciones que le ofrece la aplicación. El nombre de estos elementos es el propio de los lenguajes de programación utilizados (HTML, JavaScript y CSS).

- <IMG> → Nos sirve para situar imágenes en la página, como el logotipo de la aplicación. Sitúandola entre las etiquetas <A></A> podemos hacer que la imagen se comporte como un hiperenlace, que es lo que queremos conseguir cuando el usuario pulse en la imagen del ojo de un elemento del listado, para poder visualizarlo en el visor.

Figura 5.5. Imagen de "Ver"

- <DIV></DIV> → Nos permite crear las cuatro capas que observamos en la figura 6.4.

- Ventana *alert* de aviso → como su nombre indica, utilizaremos estas ventanas para informar al usuario de alguna acción que no desembocará en el resultado esperado, como por ejemplo visualizar las cámaras de un cliente que no tiene cámaras registradas ni configuradas.

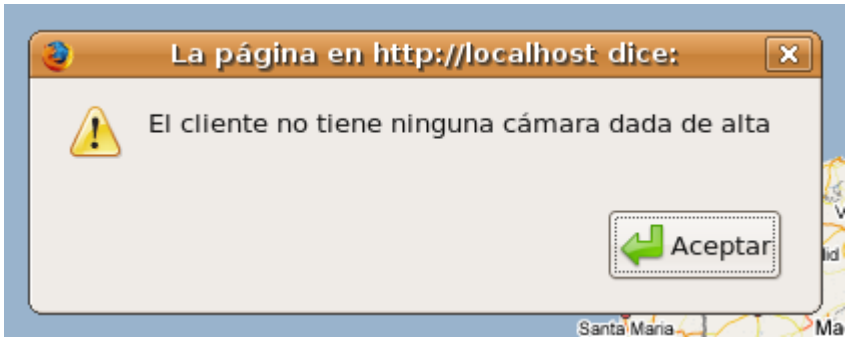

Figura 5.6. Ventana de alerta

- Visor Google Maps → creamos el mapa en la etiqueta identificada como id=”map”, gracias al constructor que facilita la API de Google Maps. Además, incluimos los siguientes controles propios de dicha API: GLargeMapControl, GScaleControl y GMapTypeControl. Aparte de estos controles, incluimos objetos como GMarker, GIcon, GPolyline y GInfoWindowHtml, y tratamiento para los eventos que se puedan producir sobre dichos elementos. Los controles y objetos de la API de Google Maps que acabamos de nombrar se han descrito en el apartado 5.2 de este documento.

6 Implementación de la aplicación web

6.1 Entorno de desarrollo

En este apartado se enumera el software que es necesario instalar para poder desarrollar e implementar la aplicación web.

En primer lugar hay que tener en cuenta que el Sistema Operativo sobre el que se va a trabajar es Ubuntu 7.04 – *Feysty Fawn* –.

Nuestra solución software es una aplicación web, que como cualquier aplicación de este tipo, tiene como primer requisito para poder ejecutarse la presencia de un servidor web. En nuestro caso usaremos Apache como servidor web o HTTP, y su labor principal será analizar cualquier archivo solicitado por un navegador y mostrar los resultados correctos en función del código del archivo.

Como lenguaje de secuencias de comandos del lado del servidor que permite que el sitio web sea realmente dinámico, utilizaremos PHP. El código PHP lo interpretará el servidor web Apache, que se encargará pues de generar código HTML y otro contenido que el usuario no verá.

El SGBDR que usaremos será MySQL, ya que aunque PHP disponga de conexión propia a todos los sistemas de bases de datos, MySQL ofrece compatibilidad con PHP y está perfectamente integrado. De hecho, PHP cuenta con una serie de funciones específicas diseñadas para interactuar con MySQL.

Para instalar Apache, PHP y MySQL podemos descargarnos cada paquete por separado e ir configurándolo, que sería lo más adecuado para la aplicación en caso de ponerla en un servidor de producción, o bien descargarnos una distribución de Apache que contenga MySQL y PHP, como XAMPP (http://www.apachefriends.org/es/xampp.html), en el caso de que vayamos a usarlo para desarrollar la aplicación y no tengamos problemas de seguridad.

La filosofía de XAMPP es construir una versión fácil de instalar para los desarrolladores que entran al mundo de Apache. Para hacerlo más conveniente para los desarrolladores, XAMPP está configurado con todas las opciones activadas. Esta configuración por defecto no es buena desde el punto de vista de la seguridad y no es suficientemente segura para un ambiente de producción. Además, XAMPP es una compilación de software libre, y también es gratuito y libre conforme a los términos de la licencia GNU.

Nosotros hemos optado por decargarnos el módulo LAMPP (XAMPP para Linux). Su instalación es

sencilla, ya que tan sólo hemos de descomprimir el contenido del archivo comprimido que descargamos en el directorio `/opt`. Para iniciarlo, simplemente escribimos en la consola, como root, `/opt/lampp/lampp start`.

Para administrar la base de datos MySQL utilizaremos la herramienta de código libre *phpMyAdmin* (http://www.phpmyadmin.net), que nos proporciona una interfaz web desde donde interactuar con nuestra base de datos.

Como entorno de desarrollo usaremos Eclipse SDK(http://www.eclipse.org/), en su versión 3.1.2. No usaremos la última versión de este popular IDE ya que también instalaremos el plugin PHPEclipse, que proporciona herramientas para convertir el framework Eclipse en un IDE para desarrolladores PHP. La última versión del plugin PHPEclipse funciona correctamente para la versión 3.1.2 de Eclipse, y no para otras más recientes como la 3.2. Para instalarlo, se puede hacer fácilmente desde el propio Eclipse.

6.2 Aspectos generales

A continuación se muestran características generales de la implementación de la aplicación web.

6.2.1 Estructura de directorios

Nuestra aplicación web sigue la siguiente estructura de directorios:

- Directorio */css*: contendrá la hoja de estilos CSS que se aplicaremos a la interfaz.
- Directorio */db*: contiene el script SQL de creación de la estructura de la base de datos de la aplicación, así como los datos ejemplo con que se cargarán las diferentes tablas.
- Directorio */images*: contiene las imágenes e iconos que usamos en la interfaz.
- Directorio */includes*:
  - *config.php* : permite al usuario introducir los datos de definición de acceso a la base de datos donde accederá la aplicación.
  - *dbconnect.php* : contiene la cadena de conexión a la base de datos. Los parámetros de conexión los toma del fichero *config.php*.
- Directorio / (directorio base): contiene el fichero *maps.php*, que es el fichero principal de la aplicación, y los scripts PHP que exportan los datos de la base de datos en formato XML que podemos recuperar y mostrar en el visor Google Maps mediante llamadas JavaScript asíncronas.

La siguiente imagen capturada del *Package Explorer* del IDE Eclipse muestra la estructura de directorios y los ficheros de que se compone la aplicación web:

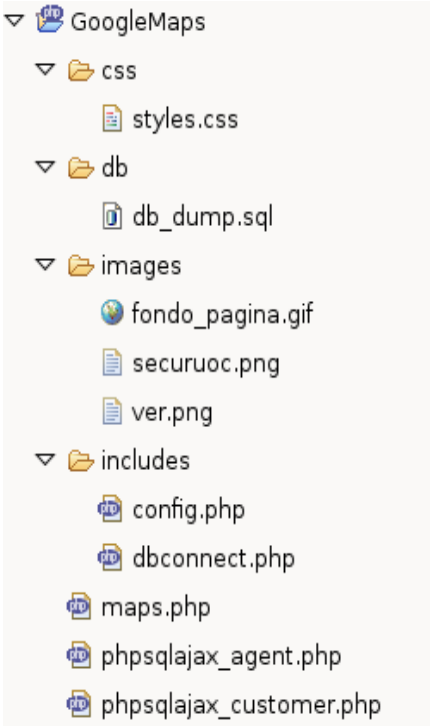

Figura 6.1. Estructura de directorios de la aplicación web

6.2.2 Código común

En la zona de código JavaScript del fichero *maps.php*, podemos observar al principio una declaración de variables globales, que utilizarán varias funciones JavaScript de nuestra aplicación y cuyo valor no se limitará al cuerpo de dichas funciones. A continuación se muestra el código comentando la función de dichas variables.

```
var map; // Variable que contendrá el objeto GMap2 (el visor de Google Maps)
var geocoder = new GClientGeocoder();  // Objeto GClientGeocoder, que permite convertir
    direcciones en objetos GLatLng
var infoTabs = new Array(); // Array de GInfoWindowTabs para mostrar la información y
    las cámaras de un cliente
var cameras; // Variable donde almacenamos los elementos <camera></camera> de un
    customer que obtenemos mediante AJAX
var routePoints = new Array(); // Array de GLatLng para mostrar la ruta de un agente
var marker; // Variable donde almacenamos el objeto GMarker asociado al cliente o al
    agente seleccionado
```

Al cargarse la página, lo primero que ocurre es que se llama a la función **load**(). En esta función, se inicializa el mapa, si el navegador es compatible, en la capa identificada como map. Se añaden los controles GLargeMapControl, GScaleControl y GMapTypeControl, y se centra la vista en el centro geográfico de la península ibérica, a un nivel de zoom de 5.

```
function load() {
       if (GBrowserIsCompatible()) {
          map = new GMap2(document.getElementById("map"));
          map.addControl(new GLargeMapControl());
          map.addControl(new GScaleControl(), new
                         GControlPosition(G_ANCHOR_BOTTOM_RIGHT, new GSize(20,20)));
          map.addControl(new GMapTypeControl());
          map.setCenter(new GLatLng(40.413496, -3.779297), 5);
        }
      }
```

Tenemos otra función que se usa tanto a la hora de mostrar los clientes como los agentes **addListenerClick**(), y recibe como parámetro un valor booleano que indica si la marca tenía una única pestaña asociada (caso de un agente) o varias (cliente con sus respectivas cámaras). Su objetivo es asociar una acción al evento de hacer *click* sobre una marca en el visor, mediante el método GEvent.addListener(), de modo que se abra la ventana InfoWindow correspondiente y, en el caso de que se pase *false* como argumento (se trata de un cliente con cámaras asociadas) también llama a la función que refresca las imágenes de las cámaras web.

```
function addListenerClick(oneTab){
       if (oneTab) {
          GEvent.addListener(marker,"click",function(){
            marker.openInfoWindowHtml(infoText);
          });
       } else {
          GEvent.addListener(marker,"click",function(){
            marker.openInfoWindowTabsHtml(infoTabs);
            refreshImages();
          });
        }
      }
```

El resto de funciones JavaScript se explican en los apartados 4.3 y 4.4, según su objetivo. No obstante, podemos mencionar el uso de la función de la API de Google Maps **GDownloadUrl()**, que se usa tanto para obtener la información de un cliente como la de un agente.

Esta función es un envoltorio para el objeto XmlHttpRequest del navegador, que se usa para solicitar un archivo XML al servidor. Por tanto, la función GDownloadUrl() proporciona una forma de recuperar en modo asíncrono un recurso identificado por su URL. El primer argumento es la ruta del script, y el segundo parámetro es la función que se invoca cuando se devuelve el XML a JavaScript.

El uso que hacemos de GDownloadUrl() se explica en detalle en los dos apartados siguientes.

6.3 Integración al visor de Google Maps de videocámaras

En este apartado se exponen las decisiones tomadas para integrar en el visor de Google Maps las videocámaras de los clientes, y se explican las funciones y el código implementados a tal efecto.

6.3.1 Script phpsqlajax_customer.php

El script *phpsqlajax_customer.php* se ejecuta en el servidor, y se encarga de devolver en formato XML la información asociada a un cliente, cuyo id que se pasa como parámetro adjuntado al final de la url (método GET).

Lo primero que debe hacer es conectarse a la base de datos, por lo que se incluye el fichero de conexión a la misma.

A continuación, vemos que se declara la función **parseToXML**(). Esta función auxiliar sirve para formatear caracteres especiales a XML (<,>,',",&).

```
include ("includes/dbconnect.php");

  function parseToXML($htmlStr)
  {
    $xmlStr=str_replace('<','&lt;',$htmlStr);
    $xmlStr=str_replace('>','&gt;',$xmlStr);
    $xmlStr=str_replace('"','&quot;',$xmlStr);
    $xmlStr=str_replace("'",'&apos;',$xmlStr);
    $xmlStr=str_replace("&",'&amp;',$xmlStr);
    return $xmlStr;
  }
```

Vemos que ahora se obtiene el valor del id del cliente que se pasa por la url. Una vez conocemos el cliente que ha solicitado el usuario, podemos generar la consulta SQL para obtener su información de la base de datos. Si no hay resultados, salimos.

```
  $idCustomer = trim($_GET['idCustomer']);

  $sql = "SELECT * FROM tCustomer WHERE idCustomer = " . $idCustomer;
  $result = mysql_query($sql);
  if (!$result) {
    die('Invalid query: ' . mysql_error());
  }
```

Después ya se empieza a generar el texto XML que hay que devolver. Empezamos especificando que el tipo devuelto será text/xml, y a continuación escribimos el elemento raíz del documento, que será *customer*. Recuperamos la fila de la base de datos y escribimos los elementos *companyName* y *address,* pasando primero los valores que recuperamos desde la BD por la función parseToXML.

```
header("Content-type: text/xml");

// Start XML file, echo parent node
echo '<customer>';

$row = @mysql_fetch_assoc($result);
//ADD TO XML DOCUMENT NODE
echo '<companyName>';
echo parseToXML($row['txCompanyName']);
echo '</companyName>';
echo '<address>';
echo parseToXML($row['txAddress']) . ', ' . parseToXML($row['txCity']);
echo '</address>';
```

Una vez tenemos la información de la tabla *tCustomer*, hemos de recuperar las cámaras asociadas. Para ello, repetimos el proceso accediendo esta vez a la tabla *tCamera*, y, mientras vayamos obteniendo datos, escribimos elementos *camera* con sus atributos *name* y *url* correspondientes. Finalmente, se cierra el elemento *customer* raíz.

```
$sql2 = "SELECT * FROM tCamera WHERE fkCustomer = " . $idCustomer;
$result2 = mysql_query($sql2);
//Iterate through the rows, adding XML nodes for each
while ($row2 = @mysql_fetch_assoc($result2)){
  echo '<camera ';
  echo 'name="' . parseToXML($row2['txName']) . '" ';
  echo 'url="' . parseToXML($row2['txURL']) . '" ';
  echo '/>';
}

echo '</customer>';
```

6.3.2 Función showCustomer(idCustomer)

Esta función, definida en el fichero principal de la aplicación (*maps.php*), es la que se encarga de llamar al script php del servidor para que le devuelva la información de un cliente, y después la muestra en el visor de Google Maps.

En primer lugar, veamos cómo realiza la petición asíncrona de la información XML mediante la función **GDownloadUrl()**:

- Se construye en la variable *query* la petición que se enviará como primer argumento de la función GDownloadUrl(), y que llevará como parámetro el *idCustomer* correspondiente.
- Después, se invoca la función GDownloadUrl(), y, si se recibe información, se formatea a XML y se guarda en la variable *responseDoc*. Hay que tener en cuenta que en nuestro caso, dado que en el script *phpsqlajax_customer.php* ya incluimos la cabecera diciendo que el formato devuelto sería text/xml, no sería necesario llamar al método Gxml.parse().
- A continuación, obtenemos el elemento raíz del documento, y lo guardamos en *root*.
- Ahora, por cada elemento que queremos recuperar, vamos llamando al método getElementsByTagName() (de [0], puesto que sólo existe un elemento *companyName* y un elemento *address*) y lo guardamos en las variables correspondientes, para después obtener el valor mediante el método Gxml.value().
- Una vez hemos recuperado el valor de los elementos *companyName* y *address*, le damos el formato HTML adecuado y lo guardamos en un objeto GInfoWindowTab, añadiéndolo al array de objetos de este tipo.
- Por último, para las cámaras, dado que pueden existir varias, una vez hemos obtenido el array con todos los elementos en una variable *camera*, vemos cuál es su longitud con el parámetro *length*. Si el cliente no tiene cámaras asociadas (longitud 0), mostraremos una alerta indicándolo; si tiene al menos una cámara (longitud mayor que 0), recorreremos cada uno de los subelementos [i] obteniendo los valores de los atributos *name* y *url*. Del mismo modo que antes, le damos formato HTML y añadimos un nuevo objeto GInfoWindowTab al array de pestañas.

```
var query = "phpsqlajax_customer.php";
query += "?idCustomer=" + idCustomer;

GDownloadUrl(query, function(data, status) {
  if (status == 200) {
    var responseDoc = Gxml.parse(data);
    var root = responseDoc.documentElement;
    var companyNameNode = root.getElementsByTagName("companyName")[0];
    companyName = Gxml.value(companyNameNode);
    var addressNode = root.getElementsByTagName("address")[0];
    address = Gxml.value(addressNode);
    var infoText = "<b> Cliente: </b> " + companyName;
    infoText += "<br> <b> Dirección: </b> " + address;
    var tab = new GinfoWindowTab("Info",infoText);
    infoTabs = [];
    infoTabs.push(tab);
    cameras = root.getElementsByTagName("camera");
    if(cameras.length <= 0) {
      alert("El cliente no tiene ninguna cámara dada de alta");
    } else {
      for (var i = 0; i < cameras.length; i++) {
        var name = cameras[i].getAttribute("name");
        var url = cameras[i].getAttribute("url");
        var imw = 300;
        var imh = 200;
        infoText = '<img name="popupCam_' + i +'" border="0" src="' + url + '"
                    width="' + imw + '" /> <br/>';
        if (i==0) {
          infoText = '<div id="cam" style="width:' + cameras.length*200 + 'px">' +
                      infoText + '</div>';
        }
        tab = new GinfoWindowTab(name,infoText);
        infoTabs.push(tab);
      }
      setTimeout("refreshImages()", 5000);
    }
```

Una vez se ha obtenido esta información mediante el motor de AJAX, utilizamos métodos propios de la API de Google Maps para mostrar la información en el visor. Sin salir aún de la función GDownloadUrl(), hemos de convertir ahora la dirección del cliente a coordenadas geográficas, ya que dicha dirección se almacena en la base de datos en formato de texto. Para ello, se hace uso del método geocoder.getLatLng(), que ya se explicó en el apartado 5.2.6.2 de la PEC3. Si la dirección existe, se sitúa sobre ella una marca y se abren las InfoWindowTabsHtml sobre la misma. Además, se añade un *listener* para escuchar y manejar eventos de *click* sobre la marca (se llama a la función addListenerClick con el valor *false*, que se ocupa de asociar el evento a la marca).

```
if (geocoder) {
  geocoder.getLatLng(address, function (point) {
    if(!point){ // Si no se puede convertir
      alert("Dirección no encontrada");
      map.setCenter(new GLatLng(40.413496, -3.779297), 5);
    } else {
      map.setCenter(point,16);
      map.clearOverlays();
      marker = new GMarker(point);
      map.addOverlay(marker);
      marker.openInfoWindowTabsHtml(infoTabs);
      addListenerClick(false);
    }
  });
}
```

Para entender mejor el funcionamiento del motor de AJAX desde que el usuario de la aplicación selecciona un cliente del listado hasta que se muestra la información asociada al cliente en el visor, se adjunta la figura 7.2.

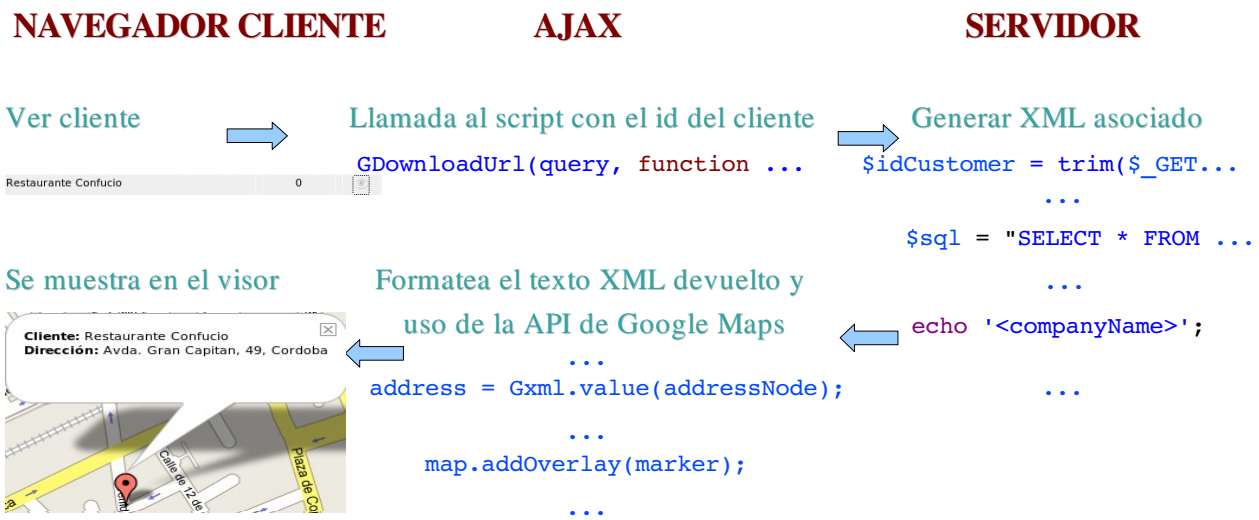

Figura 6.2. Funcionamiento motor AJAX showCustomer()

6.3.3 Funciones para actualizar la imagen de las cámaras

Como ya hemos comentado, en el visor de Google Maps lo que hacemos es mostrar mediante un objeto GInfoWindowTabsHtml las diferentes cámaras que tiene contratadas un cliente. Cada cámara se muestra en una pestaña diferente.

Ahora bien, lo que se muestra es una imagen de una cámara que está en una url en Internet. Por tanto, esta imagen es estática, no se actualiza. Para actualizarla, hacemos uso de dos funciones.

La primera función se llama **refreshImages**(), y no recibe ningún parámetro. Esta función usa la variable global *cameras*, cuyo atributo *length* nos permite conocer el número de cámaras y, consiguientemente, de imágenes que se están visualizando en el visor. Se basa en iterar, mediante un bucle, por cada una de estas imágenes que formarán parte del array *images* de JavaScript, pero sólo en las imágenes de las cámaras, refrescándolas. Para ello las hemos identificado en el código HTML de las ventanas de información concatenando 'popupCam_' con el número de cámara que correspondiera. En cada iteración del bucle, pues, se llama a la función noCache() pasándole la propiedad *src* del objeto *img* actual, y el valor que devuelve esta función será el nuevo valor de *img.src*.

La segunda función, como hemos mencionado antes, se llama **noCache**(), y recibe como parámetro la propiedad *src* de un objeto *img*. Se encarga de devolver la ruta de la imagen correspondiente a la fecha y hora actuales, para actualizar la imagen que invoca dicha función.

```
function refreshImages(){
  for (var i = 0; i < cameras.length; i++) {
    var name = 'popupCam_' + i;
    var img = document.images[name];
    if(img) {
      img.src = nocache(img.src);
    } else {
      return;
    }
  }
  setTimeout("refreshImages()",2000);
}

function nocache(src){
  now=new Date();
  if(src.indexOf("?")==-1){
    return src+"?"+now.getTime();
  } else {
    return src+"&"+now.getTime();
  }
}
```

6.4 Integración al visor de Google Maps de elementos móviles

En este apartado se explica cómo se ha resuelto la integración del visor de Google Maps con los agentes móviles, mostrando la situación actual del agente en el mapa y la ruta de vigilancia que sigue.

La estructura de la solución se asemeja a la vista en el apartado 4.3 para integrar las cámaras de los clientes en el visor, pero, en este caso, en lugar de recuperar el motor de AJAX información de las cámaras, recuperará los puntos de la ruta de un agente y el punto actual donde se encuentra desde la base de datos. A continuación se explica en detalle.

6.4.1 Script phpsqlajax_agent.php

El script *phpsqlajax_customer.php* se ejecuta en el servidor, y se encarga de devolver en formato XML la información de un agente y los puntos de la ruta que vigila (si la tiene asignada), con la posición

donde se encuentra actualmente.

Su código y funcionamiento es bastante similar al del script *phpsqlajax_customer.php*, explicado en el punto 4.3.1, por lo que únicamente nos centraremos en explicar cómo se simula y se va modificando la posición actual de un agente.

En primer lugar configuramos la consulta SQL para para recuperar los puntos de la ruta del agente, ordenados por el orden que tienen en el recorrido.

Utilizamos una variable $orden que se irá incrementando en cada iteración del bucle y que tendrá el mismo valor que el orden que tiene cada punto en el recorrido.

```
$sql3 = "SELECT * FROM tRoutePoint WHERE fkRoute = " . $route . " ORDER BY iOrden";
$result3 = mysql_query($sql3);
$numPoints = mysql_num_rows($result3);
$orden = 0;

//Iterate through the rows, adding XML nodes for each
while ($row3 = @mysql_fetch_assoc($result3)){
  $isCurrentPosition = $row3['bIsCurrentPosition'];
  $orden++;
  echo '<point ';
  echo 'lat="' . parseToXML($row3['fLat']) . '" ';
  echo 'long="' . parseToXML($row3['fLong']) . '" ';
  echo 'isCurrentPosition="' . parseToXML($isCurrentPosition) . '" ';
  echo '/>';
```

En cada iteración vamos recuperando el valor de *bIsCurrentPosition*, que nos indica si el punto actual es donde se encuentra el agente. Si es así, hemos de actualizar la fila de la base de datos poniendo el valor del atributo *bIsCurrentPosition* a 0. Además, hemos de establecer el nuevo punto donde se situará el agente. Para ello, hay dos posibilidades:

- Si no estábamos en el último punto del recorrido, la nueva posición actual del agente será el siguiente punto del mismo ($orden + 1). Configuramos la consulta para actualizar el atributo bIsCurrentPosition de este punto a 1.
- Si estábamos en el último punto del recorrido, la nueva posición actual del agente será el punto de orden 1 del recorrido. Configuramos la sentencia SQL para establecer a 1 el valor del atributo bIsCurrentPosition de este punto.

Finalmente, ejecutamos la consulta y actualizamos la BD con la nueva posición del agente.

```
    if ($isCurrentPosition) {
      $sql4 = "UPDATE tRoutePoint SET bIsCurrentPosition = 0 WHERE fkRoute = " . $route
               . " AND iOrden = " . $orden;
      mysql_query($sql4);
      if (($orden + 1) <= $numPoints) {
      $sql5 = "UPDATE tRoutePoint SET bIsCurrentPosition = 1 WHERE fkRoute = " . $route
               . " AND iOrden = " . ($orden+1);
      } else {
        $sql5 = "UPDATE tRoutePoint SET bIsCurrentPosition = 1 WHERE fkRoute = " .
                 $route . " AND iOrden = 1";
      }
        mysql_query($sql5);
    }
}
echo '</customer>';
```

Si nos fijamos, cada vez que el usuario le de al botón de "ver" sobre un agente en el listado, se ejecuta el script *phpsqlajax_agent.php*, y se actualiza la posición del agente al siguiente punto del recorrido. Otra posibilidad podría haber sido crear un script con un *timer* que actualizara la posición automáticamente cada cierto tiempo, o bien podríamos haber generado la nueva posición al azar en tiempo de ejecución.

6.4.2 Función showAgent(idAgent)

La función JavaScript *showAgent(idAgent)*, definida en el fichero principal de la aplicación (*maps.php*), se comunica con el script php del servidor en modo asíncrono para recuperar la información de un agente y los puntos de su ruta, mostrando el recorrido y la información en el visor de Google Maps.

La petición asíncrona de la información XML la obtiene realizando una petición al script del servidor *phpsqlajax_agent.php*, por medio de la función GDownloadUrl(), de modo similar a como se explica en el apartado 4.3.2 para el caso de los clientes y las cámaras:

- En la variable *query* se escribe la url del script php del servidor, concatenando con un '?' el parámetro *idAgent* y su valor, que se pasan mediante el método GET a dicho script.

- Se hace uso de la función GDownloadUrl(), que se encarga de comunicarse en modo asíncrono con el servidor utilizando el objeto XmlHttpRequest del navegador. Si se obtiene información del servidor, se formatea a XML.
- Se obtiene el elemento raíz del documento (y todo su contenido o subelementos), almacenándolo en la variable *root*.
- Recuperamos los elementos que son únicos, como *agentName* o *mobilePhone*, y creamos una cadena HTML para almacenar la información lista para mostrarla luego en el visor.
- Limpiamos las posibles marcas que pudiera haber en el visor y vaciamos el array de puntos *routePoints*.
- Por cada elemento *point*, vamos obteniendo sus atributos *lat* y *long*, que son las coordenadas geográficas del mismo. Dado que todos los datos devueltos por el XML se consideran cadenas de caracteres, necesitamos convertir la latitud y la longitud a coma flotante usando el método parseFloat(), y después creamos un objeto GLatLng. Este nuevo objeto GlatLng lo insertamos en el array de coordenadas *routePoints*.
- Por cada elementos *point*, recuperamos también el atributo *isCurrentPosition*. Si el valor de este atributo es 1, entonces este punto es la posición donde se encuentra actualmente el agente, y hemos de situar una marca con la información del agente sobre ella. También llamamos a la función *addListenerClick()* pasándole como argumento *true*, dado que en este caso la marca sólo tiene una pestaña informativa.
- Finalmente, una vez hemos terminado de recorrer los elementos *point*, ya tenemos todas las coordenadas de los puntos de la ruta del agente en el array *routePoints*, por lo que dibujamos en el visor de Google Maps el recorrido del agente por estos puntos mediante el objeto GPolyline.

```
var query = "phpsqlajax_agent.php";
query += "?idAgent=" + idAgent;

GDownloadUrl(query, function(data, status) {
  if (status == 200) {
    var responseDoc = GXml.parse(data);
    var root = responseDoc.documentElement;
    var agentNameNode = root.getElementsByTagName("name")[0];
    agentName = GXml.value(agentNameNode);
    var mobilePhoneNode = root.getElementsByTagName("mobilePhone")[0];
    mobilePhone = GXml.value(mobilePhoneNode);
    var infoText = "<b> Agente: </b> " + agentName + "<br> <b> Móvil: </b> " +
                   mobilePhone;
    map.clearOverlays();
    routePoints = [];
    points = root.getElementsByTagName("point");
    for ( var i = 0; i < points.length; i++) {
      var lat = parseFloat(points[i].getAttribute("lat"));
      var long = parseFloat(points[i].getAttribute("long"));
      var point = new GLatLng(lat, long);
      routePoints.push(point);
      var isCurrentPosition = points[i].getAttribute("isCurrentPosition");
      if (isCurrentPosition == 1) {
        infoText += "<br> <b> Latitud: </b>" + lat + " <b> Longitud: </b>" + long;
        marker = new GMarker(point);
        map.addOverlay(marker);
        marker.openInfoWindowHtml(infoText);
        map.setCenter(point,14);
        addListenerClick(true);
      }
    }
    var polyline = new GPolyline(routePoints, "#ff0000", 7);
    map.addOverlay(polyline);
  }
});
```

En la siguiente figura 7.7 se muestra el paso de información desde el navegador al servidor pasando por el motor de AJAX, y viceversa.

Figura 6.3. Funcionamiento motor AJAX showAgent()

7 Glosario

- **AJAX**: *Asynchronous JavaScript And Xml*, es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla

- **API**: *Application Programming Interface*, es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción

- **Base de datos**: una base de datos es una colección estructurada de datos y forma parte de un sistema de información

- **CAD**: *Computer Aided Design*, Diseño Asistido por Ordenador. Hace referencia al conjunto de herramientas informáticas que facilitan el diseño gráfico de elementos

- **Cartografía**: la cartografía es la ciencia que trata la representación de la Tierra sobre un mapa. Como la tierra es esférica ha de valerse de un sistema de proyecciones para pasar la esfera al plano

- **Cobertura**: una cobertura es la forma de agrupar objetos de características similares en un SIG

- **Formato ráster**: formato de intercambio de información geográfica. En este caso todos los elementos se representan de la misma manera, mediante celdas que forman una malla (de hecho ráster significa cuadrícula o malla en inglés)

- **Formato vectorial**: formato de intercambio de información geográfica. En el formato vectorial se intenta aproximar los objetos existentes mediante componentes vectoriales (puntos, líneas y polígonos)

- **GML**: *Geography Markup Language*, Lenguaje de Marcado Geográfico. Lenguaje que deriva del XML y que sirve para el modelaje, transporte y almacenamiento de información geográfica. Es el estándar a través del que se transmiten las órdenes WFS

- **Google Earth**: es un programa informático similar a un SIG, y que permite visualizar imágenes 3D del planeta, combinando imágenes satélite, mapas y el motor de búsqueda de Google

- **Google Maps**: es el nombre de un servicio gratuito de Google. Es un servidor de aplicaciones de mapas en web. Ofrece imágenes de mapas desplazables, así como fotos satelitales del mundo e incluso la ruta entre diferentes ubicaciones. Asimismo, ofrece la posibilidad de que cualquier propietario de una página web integre muchas de sus características a su sitio web

- **GPS**: *Global Positioning System*, Sistema de Posicionamiento Global. Originalmente llamado NAVSTAR, es un Sistema Global de Navegación por Satélite (GNSS), que permite determinar en todo momento la posición de un objeto, con una desviación de 4 metros

- **Latitud**: distancia angular, medida sobre un meridiano, entre una localización terrestre (o de cualquier otro planeta) y el ecuador

- **Longitud**: distancia angular horizontal, paralela al ecuador, entre el meridiano de Greenwich en Londres y un determinado punto de la Tierra

- **Meridiano**: círculo máximo que proporciona la longitud o la distancia Este-Oeste de la Tierra

- **OGC**: *Open Geospatial Consortium*. Su fin es la definición de estándares abiertos e interoperables dentro de los SIG

- **Paralelo**: línea resultante de la intersección de un plano paralelo al del Ecuador y la superficie terrestre

- **Píxel**: es el acrónimo de la palabra inglesa "pictorial element". Un píxel es cada uno de los puntos que son asignables o direccionables en una pantalla de vídeo

- **SGBD**: *Sistema Gestor de Bases de Datos*. Es un tipo de software muy específico, dedicado a servir de interfaz entre las bases de datos y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta

- **SIG**: *Sistema de Información Geográfica*, GIS en inglés. Sistema de hardware, software, información espacial y procedimientos informáticos que permiten y facilitan el análisis, gestión y representación del espacio

- **Sistemas de coordenadas**: los sistemas de coordenadas son las herramientas que permiten localizar un punto sobre la superficie terrestre

- **UML**: *Unified Modeling Language*, es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software

- **WFS**: *Web Feature Service*, servicio web definido por el OGC, que ofrece una interfaz de comunicación que permite interactuar con los mapas servidos por el estándar WMS, como por ejemplo, analizar la imagen que nos ofrece el servicio WMS siguiendo criterios geográficos.

- **WMS**: *Web Map Service*, servicio web definido por el OGC, produce mapas de datos espaciales referidos de forma dinámica a partir de información geográfica. Este estándar internacional define un "mapa" como una representación de la información geográfica en forma de un archivo de imagen digital conveniente para la exhibición en una pantalla de ordenador

8 Bibliografía

[02_01] Aronoff, S. Otawa, Ed. WDL. PUBLICATIONS, 1989.
Geographic information systems: A management perspective.

[02_02] Wikipedia. Enciclopedia mundial de acceso libre
http://es.wikipedia.org/wiki/Sistema_de_Informaci%C3%B3n_Geogr%C3%A1fica_
Último acceso: 27 marzo 2007

[02_03] Kingston Center for GIS. Introducing GIS.
http://www.emich.edu/visit/Publications/Intro_to_GIS.ppt
Último acceso: 27 marzo 2007

[02_04] Centro documental Monografías.com
http://www.monografias.com/trabajos/gis/gis.shtml
Último acceso: 27 marzo 2007

[02_05] Artículo introductorio para entender las bases de los SIG
http://recursos.gabrielortiz.com/index.asp?Info=012
Último acceso: 16 abril 2007

[02_06] Artículo sobre Software SIG
http://www.turismo.uma.es/alumnos/arcinfo/Tema_2.html
Último acceso: 27 marzo 2007

[02_07] Burrough, Peter A. y Mc. Donnell, Rachel A., University Press, Oxford, 1998
Principles of Geographical Information Systems

[03_01] WebSite del Open Geospatial Consortium (OGC)
http://www.opengeospatial.org
Último acceso: 29 marzo 2007

[03_02] Masó, Joan y Pons, Xavier. Descargas y animaciones en el navegador de mapas OGC de Miramon.
http://www.creaf.uab.es/miramon/publicat/papers/jidee04/Article_Descargas%20y%20animaciones%20en%20el%20navegadors%20de%20mapas.pdf
Último acceso: 29 marzo 2007

[03_03] Wikipedia. Enciclopedia mundial de acceso libre
http://es.wikipedia.org/wiki/Web_Map_Service
Último acceso: 29 marzo 2007

[03_04] OpenGIS Web Map Service CookBook
http://portal.opengeospatial.org/files/?artifact_id=7769
Último acceso: 30 marzo 2007

[03_05] OpenGIS Web Map Service Implementation Specification, Version 1.1.1
http://www.opengis.org/docs/01-068r2.pdf
Último acceso: 30 marzo 2007

[03_06] Nuevo servidor de geoservicios del ICC
http://www.icc.cat/web/content/es/prof/cartografia/cartografia_geoserveis_newserver.html
Último acceso: 6 abril 2007

[03_07] Manual de integración de servicios web del ICC
http://www.icc.cat/pdf/es/prof/cartografia/geoservicios/manual_integracion_servicios_web_icc.pdf
Último acceso: 6 abril 2007

[03_08] GeoConnections – Developer's Corner – Technical Architecture – Web Feature Service (WFS)
http://www.geoconnections.org/CGDI.cfm/fuseaction/technical.web_feature_service/gcs.cfm
Último acceso: 31 marzo 2007

[03_09] OGC WFS 1.1.0 Implementation Specification
http://www.geoconnections.org/CGDI.cfm/fuseaction/technical.web_feature_service/gcs.cfm
Último acceso: 31 marzo 2007

[03_10] Descripción del servidor de mapas – Atlas virtual de aves terrestres de España
http://161.111.161.171/Atlas/descripcion.php
Último acceso: 1 abril 2007

[04_01] Wikipedia. Enciclopedia mundial de acceso libre
http://es.wikipedia.org/wiki/Google_Maps
Último acceso: 8 abril 2007

[04_02] AJAX: a new approach to web applications – Jesse James Garret
http://adaptivepath.com/publications/essays/archives/000385.php
Último acceso: 8 abril 2007

[04_03] Google Maps API, V2 – Scott Davis
http://www.pragmaticprogrammer.com/titles/sdgmapi/
Último acceso: 9 abril 2007

[04_04] Artículo – Mostrar capas WMS sobre Google Maps

http://georeferencias.wordpress.com/2006/08/22/mostrando-capas-wms-sobre-google-maps/
Último acceso: 12 abril 2007

[04_05] Wikiloc: software libre y APIs de Google Maps para visualizar y compartir rutas GPS
http://www.sigte.udg.es/JornadasSIGLibre/comun/1pdf/3.pdf
Último acceso: 15 abril 2007

[05_01] Pressman, Roger S., Ed. McGraw Hill, 2002.
Ingeniería del Software: Un enfoque práctico.