IG31

PROYECTOS INFORMÁTICOS DE GESTIÓN

MEJORA DE LA APLICACIÓN DE MAPAS DE GUIFI.NET

al063980@uji.es
David Rubio Miguélez

ÍNDICE GENERAL

PROPUESTA TÉCNICA..... 4
1.1. Introducción..... 5
1.2. Condiciones iniciales..... 5
1.2.1. Conceptos básicos..... 5
1.2.2. Tecnología..... 6
1.2.2.1. Drupal 6..... 6
1.2.2.2. Google Maps API v3..... 7
1.3. Objetivos..... 8
1.4. Alcance del sistema..... 9
1.4.1. Alcance funcional..... 9
1.4.2. Alcance informático..... 10
1.5. Restricciones..... 10
1.6. Alternativas..... 10
1.6.1. Migración a otras API de mapas..... 10
1.6.2. Migración a una versión más reciente de Drupal.... 11
1.6.3. Cambio de SGBD..... 11
1.7. Requisitos del sistema..... 12
1.7.1. Requisitos de datos..... 12
1.7.2. Requisitos funcionales..... 12
1.8. Tecnología..... 13
1.9. Planificación temporal..... 14
1.10. Presupuesto..... 16
ANÁLISIS DEL SISTEMA..... 17
2.1. Introducción..... 17
2.2. Diagrama de contexto..... 18
2.3. DFD. Primera explosión..... 20
2.4. DFD. Segunda explosión..... 21
2.4.1. Gestión de capas..... 21
2.4.1.1. Almacenes..... 22
2.4.1.2. Flujos de datos..... 22
2.4.1.3. Procesos..... 23
2.4.2. Gestión de nodos dinámicos..... 24
2.4.2.1. Almacenes..... 24
2.4.2.2. Flujos de datos..... 25

2.4.2.3. Procesos ... 26
2.5. Modelo conceptual de datos ... 27
2.5.1. Relaciones ... 28
2.5.2. Entidades ... 28
DISEÑO ... 31
3.1. Introducción ... 31
3.2. Diseño de interfaces ... 32
3.2.1. Panel lateral ... 33
3.2.2. Selector de capas ... 36
3.2.3. Capa de nodos dinámicos ... 40
3.2.4. Capas extra ... 43
3.3. Diseño de la base de datos ... 49
3.4. Modelo de diseño ... 49
IMPLEMENTACIÓN Y PRUEBAS ... 54
4.1. Introducción ... 54
4.2. Decisiones de implementación ... 54
4.2.1. Estructura de archivos ... 54
4.2.2. Organización de trabajo ... 55
4.2.3. Rendimiento ... 57
4.2.4. Limitaciones ... 58
4.3. Pruebas de funcionamiento ... 59
DESPLIEGUE ... 60
5.1. Introducción ... 60
5.2. Servidor de teselas de OSM ... 60
5.2.1. Instalación del software necesario ... 62
5.2.2. Configuración ... 64
5.2.3. Predibujar algunas teselas ... 69
5.2.4. Probando la instalación ... 71
ACCESO A LA APLICACIÓN ... 73
CONCLUSIONES FINALES Y TRABAJO FUTURO ... 74
APÉNDICE ... 78
8.1. Glosario ... 79
8.2. Bibliografía ... 82

CAPÍTULO 1: PROPUESTA TÉCNICA

ÍNDICE

Capítulo 1: PROPUESTA TÉCNICA 4

1.1. Introducción 5
1.2. Condiciones iniciales 5
1.2.1. Conceptos básicos 5
1.2.2. Tecnología 6
1.2.2.1. Drupal 6 6
1.2.2.2. Google Maps API v3 7
1.3. Objetivos 8
1.4. Alcance del sistema 9
1.4.1. Alcance funcional 9
1.4.2. Alcance informático 10
1.5. Restricciones 10
1.6. Alternativas 10
1.6.1. Migración a otras API de mapas 10
1.6.2. Migración a una versión más reciente de Drupal 11
1.6.3. Cambio de SGBD 11
1.7. Requisitos del sistema 12
1.7.1. Requisitos de datos 12
1.7.2. Requisitos funcionales 12
1.8. Tecnología 13
1.9. Planificación temporal 14
1.10. Presupuesto 16

1.1. INTRODUCCIÓN

El objetivo del presente documento es proporcionar un seguimiento por escrito del desarrollo de un sistema informático. Dicho documento puede usarse para conocer el estado del sistema en desarrollo y del proceso por el que ha pasado.

La primera fase en un desarrollo informático es la propuesta técnica, es decir, una especificación detallada de lo que se pretende conseguir con la realización del proyecto. Asimismo incluye también las previsiones de costes temporales y monetarios.

Puesto que este proyecto está orientado a modificar un sistema informático existente, se incluye también una descripción del mismo antes de las modificaciones (Condiciones Iniciales). Todo este apartado ha sido objeto de una revisión previa muy extensa y exhaustiva, y se incluye como tal en los costes temporales.

1.2. CONDICIONES INICIALES

El proyecto se centra en la web de gestión de la red Guifi.net. Guifi.net es una red informática abierta, libre y neutral, construida y gestionada por los propios usuarios, en la que se ofrecen servicios sin coste alguno por el uso de la red.

El portal web de Guifi.net ha sido desarrollado de forma colaborativa por una comunidad más o menos amplia de desarrolladores a lo largo de varios años y sigue siendo un proyecto de software libre en evolución.

Puesto que esta red abarca decenas de miles de ubicaciones geográficas, una parte fundamental de dicha gestión es la visualización de dichas ubicaciones geográficas, las relaciones entre ellas y el estado de las mismas.

1.2.1. CONCEPTOS BÁSICOS

Para entender apropiadamente la parte del sistema inicial que es relevante para el proyecto es necesario introducir algunos términos:

- **Nodo**: Ubicación geográfica puntual donde se halla o se prevé instalar uno o más dispositivos de red (usualmente antenas). Un usuario del sistema puede ser dueño (a efectos de gestión) de varios nodos, necesitando especificar al menos uno para poder hacer uso de la red.
- **Enlace**: Conexión por cable o inalámbrica entre dos dispositivos. Lo habitual es conectar los dispositivos de un mismo nodo por cable, y los de nodos diferentes de manera inalámbrica. A efectos de visualización en un mapa, las conexiones dentro de un mismo nodo no se consideran, por estar en la misma ubicación. Así pues, en adelante usaremos una simplificación del concepto de enlace diciendo que es una conexión entre dos nodos.
- **Nodo cliente**: Nodo que no permite conectar otros nodos cliente. En nuestro sistema, simplificaremos la definición llamando nodo cliente a aquellos nodos con uno o ningún enlace.
- **Supernodo**: Nodo que dispone de dos enlaces o más hacia otros nodos, normalmente utilizando más de una antena y colaborando en el encaminamiento dinámico.

- **Enlace troncal**: Enlace entre dos supernodos.
- **Estado de un nodo o enlace**: Los nodos y los enlaces se pueden clasificar de varias formas según su estado de funcionamiento o intención:
    - **Proyectado (“planned”)**. Éste es el estado inicial de un nodo. Representa una ubicación posible pero todavía sin instalación alguna. En el caso de los enlaces representa la intención de conectarse a otro nodo en un futuro. Su utilidad es principalmente la de ayudar a la planificación futura de la red.
    - **Reservado (“reserved”)**. Representa nodos o enlaces que solo están en funcionamiento durante determinados períodos de tiempo, pero que no obstante se les quiere asignar recursos fijos (principalmente direcciones IP).
    - **En construcción (“building”)**. Representa nodos o enlaces que están siendo instalados o configurados y pueden no ser totalmente funcionales (o nada en absoluto).
    - **En pruebas (“testing”)**. Representa nodos o enlaces que ya se han terminado de instalar y ya funcionan, pero aún están en periodo de pruebas, por lo que la configuración aún puede cambiar.
    - **Operativo (“working”)**. Representa nodos o enlaces que ya están funcionando y disponen de su configuración definitiva. Es el tipo de nodo más común.
    - **Borrado (“dropped”).** Representa nodos o enlaces que se han declarado como eliminados de la red. Este estado no es muy habitual, ya que la mayoría de usuarios, en lugar de borrar sus nodos, simplemente los dejan en desuso sin actualizar el estado en la web.

Para una recopilación más extensa de terminología se puede consultar el Glosario al final del documento.

1.2.2. TECNOLOGÍA

El sistema inicial es dependiente de una serie de tecnologías que citaremos a continuación:

1.2.2.1. DRUPAL 6

Drupal es un gestor web de contenidos modular de propósito general y libre. Está programado en PHP. Toda la parte de gestión de redes que hace Guifi.net es en realidad un módulo de Drupal llamado “Guifi”, cuyo código está alojado en Gitorious.org.

Una de las principales características de Drupal es su independencia del sistema gestor de base de datos, es decir, que podemos usar MySQL, PostgreSQL, Oracle DB, etc. sin tener que modificar las consultas.

El programador de un módulo de Drupal, si va a hacer uso de la base de datos, tiene que declarar las tablas, columnas e índices que va a usar en un array estructurado siguiendo un formato determinado llamado “Schema”. Las consultas SQL se deben ejecutar a través de unas funciones definidas en la API de Schema, de modo que antes de ser enviadas a la base de datos son modificadas automáticamente para hacerlas compatibles con el SGBD en cuestión. Otra ventaja de la API de Schema es la seguridad añadida, ya que también comprueba si la consulta es un intento de ataque por inyección SQL.

Entre las desventajas de usar esta API se encuentra el problema de que el soporte de SQL es el

mínimo común entre los diferentes SGBD (es decir, sólo tipos y funciones básicas). Entre otras cosas no soporta la creación de “triggers”, ni el uso de extensiones espaciales (tipos de datos geométricos).

Las extensiones espaciales (“Spatial Extensions” para MySQL, “PostGIS” para PostgreSQL, etc.) son extensiones no estándar al lenguaje SQL para soportar el almacenamiento de datos de carácter geométrico, como puntos, líneas, polígonos, etc., así como funciones para el cálculo de distancias, áreas o volúmenes. También dan soporte a diferentes sistemas de coordenadas (x/y, latitud/longitud, etc.) y diferentes proyecciones geodésicas (Mercator, Robinson, Mollweide, Gall-Peters, Plate carrée, etc.). Los diferentes tipos de datos espaciales son almacenados con frecuencia en un tipo de estructura llamada Árbol-R que acelera las búsquedas de elementos dentro de una región.

El soporte para extensiones espaciales ha mejorado en la versión 7 de Drupal. El módulo Guifi sigue aún en Drupal 6.

1.2.2.2. GOOGLE MAPS API V3

Las aplicaciones interactivas de visualizado de mapas que vemos frecuentemente en la web (Google Maps, visor de OpenStreetMap, etc.) se programan normalmente sobre una librería o API que simplifica la mayoría de funciones comunes, como la recuperación de mapas a través de múltiples protocolos (WMS, servidores de teselas, etc.), el control del zum, superposición de capas, etc.

Algunas de las librerías más populares son Google Maps API, OpenLayers y Leaflet.

Google Maps API es una API propietaria aunque gratuita programada por Google. Está diseñada para ser totalmente opaca, de forma que podemos añadir nuevos controles al mapa pero personalizar los existentes es casi imposible. El añadido de nuevas capas procedentes de servicios que no son de Google corre a cuenta del programador. Esto hace que sea muy fácil de usar si queremos usar solo servicios de Google, pero difícil si queremos añadir servicios externos.

OpenLayers y Leaflet son dos librerías libres y multiproveedor. Esto significa que no se han diseñado en torno a un solo proveedor de servicios, sino que permiten acceder a múltiples servicios y protocolos diferentes. Usando estas librerías no es necesario escribir código para usar diferentes protocolos, sino que solamente necesitamos configurar el tipo de servicio y la librería se ocupa del resto.

OpenLayers está programada por comunidad y trata de ser una librería lo más completa posible, mientras que LeafLet está programada por una empresa (Cloudmade, aunque también hay una comunidad detrás) y está orientada a la sencillez y consumo mínimo de recursos.

En cualquier caso, los mapas dentro de una aplicación de este tipo están compuestos por capas. Estas capas, a su vez, se componen de grupos de imágenes, estáticas (PNG, JPG) o dinámicas (gráficos vectoriales).

Las imágenes estáticas que forman una capa con datos suelen ser cuadradas y se colocan una al lado de otra. La aplicación solo pide al servidor las imágenes que son visibles, o se prevé que serán visibles pronto. Dichas imágenes son opacas si pertenecen a la capa base. Si pertenecen a cualquier otra capa deben contener alguna transparencia.

Cuando se dibuja dinámicamente sobre una capa, podemos hacerlo también con imágenes estáticas o gráficos vectoriales. Por ejemplo, suele ser costumbre colocar “pines”, que son imágenes estáticas (a menudo PNG), en los puntos interesantes del mapa.

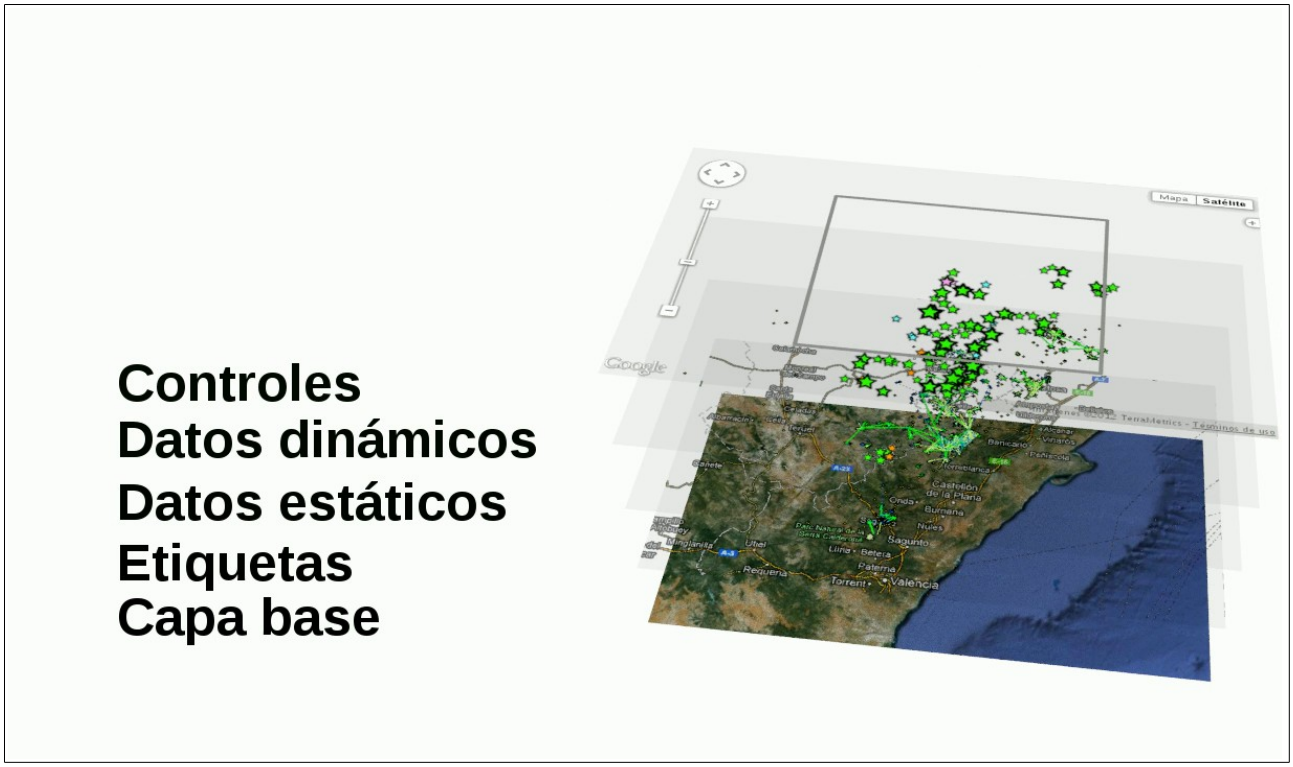


Ilustración 1: Desglose de capas dentro de una aplicación de visualizado de mapas

En la Ilustración 1 podemos ver un ejemplo de desglose de la aplicación finalizada. La capa base está compuesta de una serie de imágenes JPG cuadradas. Cualquier parte sobresaliente es cortada automáticamente por los límites del “div” contenedor. La capa de etiquetas contiene los dibujos de las carreteras, nombres de ciudades, etc. dibujados en una serie de imágenes PNG con transparencias. La capa de datos estáticos tiene la misma estructura que la anterior, pero el servidor la regenera periódicamente con datos actualizados. La capa de datos dinámicos se genera pidiendo al servidor una lista de los nodos en la región, y dibujándolos en el cliente de acuerdo a una serie de reglas, usando un formato parecido al SVG, sobre elementos “canvas”. La capa de controles es una mezcla de imágenes estáticas y dinámicas, y contiene los elementos de la interfaz para interactuar con el mapa.

1.3. OBJETIVOS

El objetivo principal del proyecto es añadir nuevas funcionalidades al apartado de mapas de Guifi.net, mejorando con ello la visualización y la gestión desde el mismo mapa de elementos de carácter geográfico, como son los nodos y los enlaces entre ellos.

El sistema debe permitir:

- **Interacción con los nodos**: Al situarse encima de un nodo, la aplicación debe mostrar información básica del nodo, y al pinchar sobre él debe redireccionar a la página de ese nodo.
- **Filtro de nodos**:
  - Ver/ocultar nodos clientes
  - Ver/ocultar el estado: proyectados/en pruebas/operativos/...

    - Ver/ocultar enlaces
    - Ver/ocultar nodos/nombres
    - Ver/ocultar zonas (los recuadros que las forman) con varios niveles de profundidad en diferentes colores.
- **Cambio de capa base**: Se debe poder cambiar la capa base por mapas de distintos proveedores.

Además de estos objetivos, que podríamos llamarlos principales, el sistema puede incluir las siguientes mejoras:

- Crear un servidor propio de mapas con los datos del proyecto OpenStreetMap para no cargar el servicio gratuito de OpenStreetMap.org. Para colaborar con este proyecto libre, el servidor podrá estar disponible tanto en Internet como en Guifi.net.
- Modificar el mapa para que permita ver el estado de la troncal en este momento (ancho de banda que ofrece un enlace y estado de "saturación" actual).
- Mejorar la aplicación del traceroute visual, de forma que se pueda mostrar la ruta real que se está utilizando en Guifi.net.

Además de los objetivos principales y las mejoras, el sistema debe cumplir con otros objetivos deseables en cualquier proyecto:

- Que sea **mantenible**. Especialmente si consideramos que el proyecto formará parte de un proyecto mayor, en el que intervienen muchas personas.
- Que sea **fiable**. Tiene que hacer las tareas para las que ha sido diseñado sin problemas.
- Que sea **eficiente**. Tiene que consumir los mínimos recursos posibles.
- Debe poseer una **interfaz sencilla**. Sobre todo si tenemos en cuenta que la aplicación la van a usar miles de personas acostumbradas a la antigua interfaz.

1.4. ALCANCE DEL SISTEMA

1.4.1. ALCANCE FUNCIONAL

Nuestro proyecto consistirá en la modificación de la aplicación actual de mapas, la cual proporcionará una serie de servicios.

El sistema:

1. Proporcionará de una manera elegante una interfaz para elegir entre proveedores de mapas.
2. Permitirá elegir entre diversas capas de cada proveedor de mapas, de una manera similar a la ya existente para los mapas de Google.
3. Permitirá activar y desactivar la visibilidad de los nodos y enlaces según distintos criterios.
4. Ofrecerá una representación visual adecuada de los nodos y enlaces según ciertas características de los mismos, como puede ser su estado o importancia.
5. Ofrecerá información básica sobre los nodos y permitirá acceder a la página de gestión de cada nodo de una forma sencilla.

1.4.2. ALCANCE INFORMÁTICO

1. Nuestro sistema informático va a apoyarse en otras partes del sistema existente para recuperar los datos de los nodos y enlaces en una región, así como información estadística sobre los mismos. No obstante dicha funcionalidad habrá de extenderse para hacerla eficiente y ajustarla al caso de uso de búsqueda de elementos dentro de una región.
2. Nuestro sistema ofrecerá dicho servicio de recuperación de datos a aplicaciones externas que lo soliciten, mediante peticiones a la web siguiendo un formato determinado, así como internamente a futuros submódulos.

1.5. RESTRICCIONES

El proyecto está sujeto a las siguientes restricciones:

- Debe realizarse dentro de los plazos establecidos y entrar dentro del número de créditos de la asignatura.
- Debe integrarse completamente en la web existente.
- Debe poder funcionar bien en un hardware similar al actual.
- No debe divergir mucho del código actual, puesto que proyectos con modificaciones extensas son menos propensos a ser integrados.

1.6. ALTERNATIVAS

Se consideraron las alternativas que se mencionan a continuación:

1.6.1. MIGRACIÓN A OTRAS API DE MAPAS

Se consideró la migración a OpenLayers o Leaflet.

Leaflet se descartó por su minimalismo, por temor a que en un futuro se hiciera necesario usar alguna función avanzada y no estuviera disponible en la API. Como ventaja habría que destacar la eficiencia. Ciertamente es una alternativa viable para un portal web externo que tuviera un subconjunto de la funcional de Guifi.net, y que hiciera peticiones de datos a la página principal. Por ejemplo, algunas regiones como el barrio de Gracia en Barcelona, tienen portales propios (GraciaSenseFils.net) en los que se muestra un mapa con el estado actual de la red en esa región, sin ninguna interacción. Para esos casos de uso, Leaflet puede ser una buena elección.

Openlayers se consideró por su potencia y amplio rango de funcionalidad, así como la extensa comunidad que lo desarrolla. OpenLayers ha demostrado ser una librería “todo terreno” capaz de conseguir todos los objetivos de este proyecto y más, con extrema facilidad y rapidez en la programación. Como parte de la evaluación de alternativas, se construyó un prototipo funcional basado en OpenLayers que consiguió alcanzar todos los objetivos principales de este proyecto, y que se muestra en la Ilustración 2:

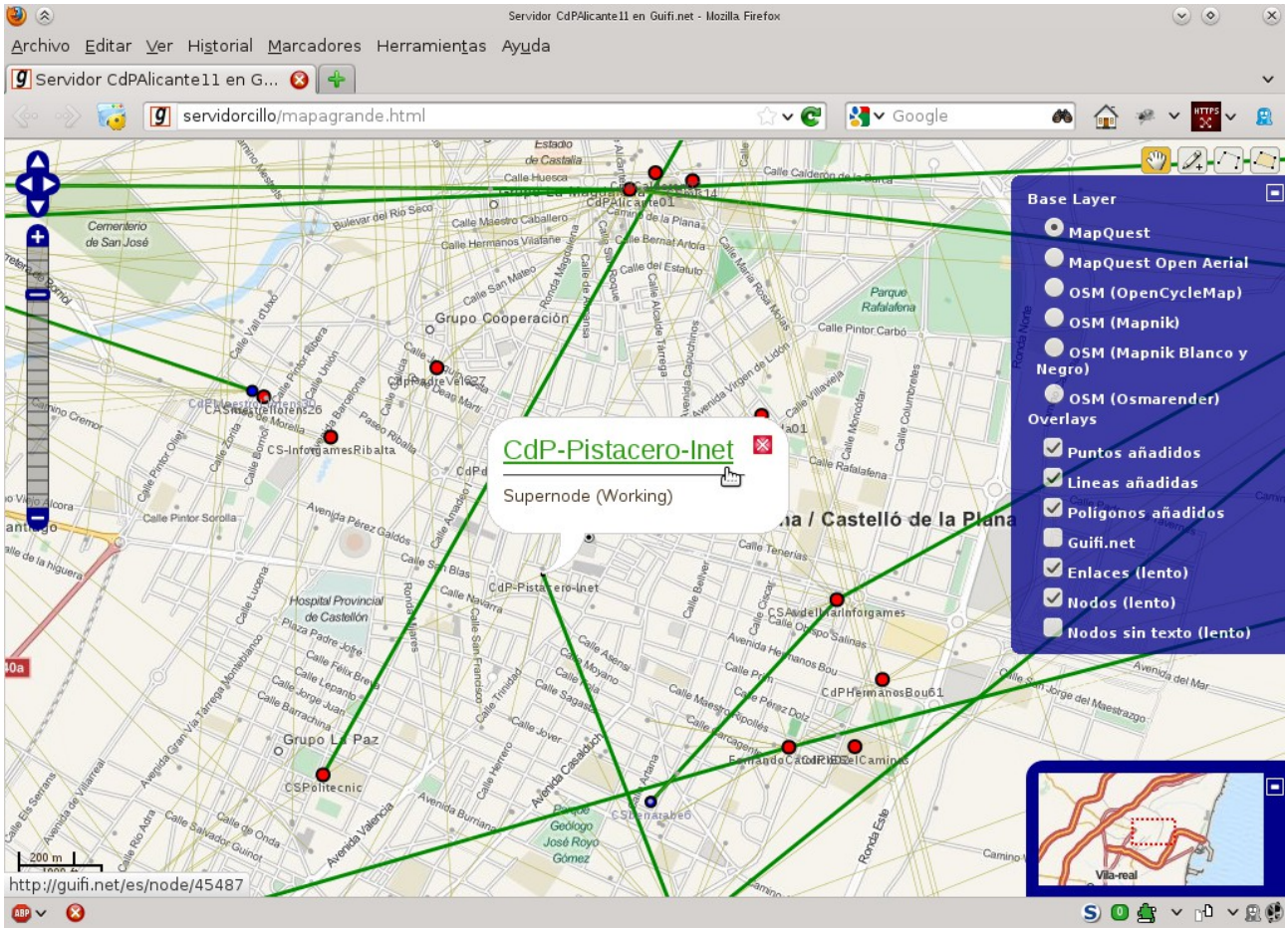


Ilustración 2: Prototipo del proyecto con la librería OpenLayers

Desgraciadamente, el prototipo no cumplía la premisa principal del proyecto: la integración con el portal Guifi.net.

Finalmente, la migración a cualquiera de estas APIs se descartó por limitaciones de recursos (principalmente tiempo), ya que una migración completa de todos los submódulos de Guifi que hacen uso de funciones de mapeado podría llevar años, y sería necesario hacer extensas pruebas antes de entrar en producción.

Así pues, se decidió continuar el desarrollo bajo la API de Google Maps V3, aunque una migración hacia OpenLayers sería la mejor solución a largo plazo.

1.6.2. MIGRACIÓN A UNA VERSIÓN MÁS RECIENTE DE DRUPAL.

Se consideró migrar todo el módulo de Guifi a Drupal 7 por tener éste un mejor soporte para tipos de datos espaciales (integrados en la API de “Schema”), pero se descartó rápidamente por la cantidad enorme de trabajo que ello supondría.

1.6.3. CAMBIO DE SGBD

Se consideró cambiar la base de datos actual de Guifi.net de MySQL a PostgreSQL, por el mejor soporte de datos espaciales, y por el mejor rendimiento. Finalmente se descartó por motivos de tiempo, y por posibles problemas no previstos.

1.7. REQUISITOS DEL SISTEMA

1.7.1. REQUISITOS DE DATOS

RD-01	Nodos
Descripción	El sistema consultará información de los nodos, que se encuentra ya almacenada en un sistema externo.
Datos específicos	• Código del nodo • Nombre del nodo • Latitud • Longitud • Estado • Estabilidad del nodo

RD-02	Enlaces
Descripción	El sistema consultará información de los enlaces, que se encuentra ya almacenada en un sistema externo.
Datos específicos	• Código del nodo asociado • Estado del enlace

RD-03	Enlaces
Descripción	El sistema obtendrá capas (conjuntos de imágenes), que se encuentran ya almacenadas en un sistema externo.
Datos específicos	• Coordenadas • Nivel de zum

1.7.2. REQUISITOS FUNCIONALES

RF-01	Mostrar nodos dentro de una región
Objetivo	El sistema debe poder mostrar un conjunto de nodos dentro de una región (especificada por la latitud y la longitud de las esquinas de la diagonal) que sean compatibles con una serie de filtros especificados.
Requisito asociado	RD-01 RD-02
Precondiciones	1. Los parámetros de latitud y longitud son correctos (están en el rango y en el orden correcto). 2. Los filtros son válidos
Secuencia normal	1. Se comprueba que los parámetros son correctos 2. Se obtiene la lista de nodos
Postcondiciones	1. Mensaje que informa que los parámetros son incorrectos

RF-02	Mostrar capa
Objetivo	El sistema debe poder mostrar capas externas.
Requisito asociado	RD-03
Precondiciones	1. Las coordenadas y el nivel de zum son válidos.
Secuencia normal	1. Se comprueba que los parámetros son correctos 2. Se obtiene la capa
Postcondiciones	1. Se devuelve una capa vacía

1.8. TECNOLOGÍA

Para la realización del presente proyecto se han usado las siguientes tecnologías:

- **Linux**, como sistema operativo de escritorio (**OpenSUSE**) y como servidor (OpenSUSE y **Debian**). El entorno de escritorio elegido para el ordenador local fue **KDE**, por mi familiaridad con él, la gestión avanzada de ventanas, y el soporte para múltiples escritorios y actividades.
- **Kate** (editor de texto avanzado), como editor de texto para editar el código fuente. Lo más útil fue el resaltado de sintaxis, la capacidad para editar ficheros a distancia (por **SSH**) de forma transparente, la gestión de sesiones, la capacidad de abrir múltiples ficheros simultáneos y la integración con la aplicación de terminal (**Konsole**).
- **MySQL**. Como base de datos para la instalación de **Drupal**, tanto para pruebas en local como en el servidor. Especialmente útil fue la función EXPLAIN para optimizar las consultas, y la utilidad "**mysqlslap**" para ejecutar una misma consulta miles de veces y medir el tiempo que tarda.
- **PHPMyAdmin**. Como editor y visor rápido de la base de datos en el ordenador local.
- **PostgreSQL** (con la extensión **PostGIS**). Como base de datos para los datos de OpenStreetMap.
- **Tilemill**. Programa para diseñar hojas de estilo para mapas. Se usó para crear un estilo adicional al estilo por defecto de OpenStreetMap optimizado para su uso en Guifi.
- **Mapnik**. Programa que realiza el dibujo ("renderizado") de un mapa a partir de los datos en la base de datos.
- **Apache**. Como servidor web para la web de Guifi. El módulo de Apache "**mod_tile**" se usó para optimizar la solicitud de teselas y su cacheado.
- **GCC**. Para compilar múltiples programas necesarios para la puesta en marcha del servidor de OpenStreetMap, incluido el módulo de Apache "**mod_tile**" y sus dependencias.
- **Mozilla Firefox** y **Google Chrome**. Como navegadores para probar el resultado. Especialmente útiles fueron las funciones de depuración de Chrome y la extensión **Firebug** de Firefox.
- **JSLint** (servicio web). Validador de código Javascript, y corrector de estilo.
- **Git**. Como sistema de control de versiones. Muy útil por las funciones de clonado, creación de ramas, mezclado, etc. También se usó el servicio web Gitorious y herramientas gráficas como **gitk** para ver la historia de versiones.

- Dropbox. Como sistema para hacer copias de seguridad automáticas para la memoria.
- **Visual Paradigm for UML 8.3 Community Edition**. Para generar los gráficos del MCD.
- **Gimp**, **Blender**, **Kdenlive**. Para componer algunas de las imágenes de esta memoria.
- **LibreOffice Writer**. Para escribir esta memoria.

1.9. PLANIFICACIÓN TEMPORAL

La planificación de un proyecto consiste en identificar las tareas para su desarrollo y estimar su duración. Cada tarea se podrá descomponer en otras.

La lista de tareas a realizar, junto con su previsión de duración es la siguiente:

- Fase de formación (105h)
  - Formación del uso de herramientas (20h)
    - Uso de Git (10h)
    - Uso de LibreOffice Writer (10h)
  - Estudio de la arquitectura del módulo Guifi (30h)
  - Estudio de la arquitectura de OpenLayers (10h)
  - Estudio de la arquitectura de Google Maps (20h)
  - Estudio de OpenStreetMap (20h)
    - Estudio del volcado en la BD de datos de OSM (10h)
    - Mapnik (2h)
    - mod_tile (4h)
    - Uso de TileMill (4h)
  - Estudio de alternativas de mapeado (5h)
    - MapProxy (4h)
    - gvSIG (1h)
- Preparación del entorno (6h)
  - Preparación del entorno en el ordenador local (4h)
    - Instalación de las herramientas (1h)
    - Instalación de Guifi en local (2h)
    - Preparación del gestor de versiones (1h)
  - Montaje del servidor de desarrollo (2h)
    - Instalación de Guifi en el servidor de desarrollo (1h)
    - Preparación del gestor de versiones (1h)
- Montaje del servidor de OpenStreetMap (10h)

- Análisis (20h)
- Diseño (30h)
  - Diseño de interfaces (10h)
  - Diseño del software (20h)
- Implementación (130h)
  - Parte del servidor (10h)
  - Parte del cliente (110h)
    - Panel lateral (30h)
    - Selector de capas (60h)
    - Nodos dinámicos (20h)
    - Otros (10h)
- Pruebas (2h)
- Escritura de la memoria (60h)
- Comunicación con profesores (10h)

Total: 373 horas.

El proyecto se ha de hacer en el transcurso de 9 meses (270 días). No obstante debemos sustraer 90 horas por periodos de exámenes, fiestas (en las cuales trabajo) y situaciones excepcionales (enfermedad, etc).

Nos quedan entonces 180 días utilizables, que corresponden con una media de 2 horas al día para hacer el proyecto, aunque en la práctica algunos días trabajaré más horas que otros.

1.10. PRESUPUESTO

Las herramientas necesarias para el desarrollo de este proyecto (ver apartado 1.8 Tecnología) son todas libres y gratuitas, con lo que el coste por herramientas es cero.

El coste de hardware se basa en la existencia de un servidor ya en funcionamiento y en la compra un disco duro corriente de 2 Terabytes de capacidad para el almacenamiento de los datos de OpenStreetMap. El espacio se descompone de la siguiente manera:

- 400 GB para el almacén de los datos procesados de OpenStreetMap en la base de datos PostgreSQL
- 500 GB para el almacén de las curvas de nivel.
- 600 GB para la caché de teselas.
- El espacio restante se deja libre para poder almacenar los archivos de datos nuevos de OSM (archivos diferenciales o “diffs”). Dichos archivos se generan periódicamente (diariamente, semanalmente, etc) y contienen solo las geometrías y datos que varían en la base de datos de OSM. Periódicamente hay que introducirlos en la base de datos si se quiere estar actualizado. También es conveniente dejar algo de espacio para mejorar el rendimiento del sistema de archivos.

El desarrollo del proyecto transcurre a lo largo de 9 meses. No obstante la duración en horas se estima en unas 373. Cada hora de desarrollo se cobra a 20 €, por lo que el importe por el tiempo invertido es de 7460€.

Recursos	**Coste**
Software	0,00 €
Hardware	80,00 €
Recursos humanos	7.460,00 €
Coste total del proyecto:	7.540,00 €

CAPÍTULO 2: ANÁLISIS DEL SISTEMA

ÍNDICE

CAPÍTULO 2: ANÁLISIS DEL SISTEMA 17

2.1. INTRODUCCIÓN 17
2.2. DIAGRAMA DE CONTEXTO 18
2.3. DFD. PRIMERA EXPLOSIÓN 20
2.4. DFD. SEGUNDA EXPLOSIÓN 21
2.4.1. GESTIÓN DE CAPAS 21
2.4.1.1. ALMACENES 22
2.4.1.2. FLUJOS DE DATOS 22
2.4.1.3. PROCESOS 23
2.4.2. GESTIÓN DE NODOS DINÁMICOS 24
2.4.2.1. ALMACENES 24
2.4.2.2. FLUJOS DE DATOS 25
2.4.2.3. PROCESOS 26
2.5. MODELO CONCEPTUAL DE DATOS 27
2.5.1. RELACIONES 28
2.5.2. ENTIDADES 28

2.1. INTRODUCCIÓN

El objetivo principal de esta fase es transformar los requisitos de la fase anterior en una posible solución al problema planteado, modelando el sistema de una manera conceptual, sin entrar en detalles de implementación, como podría ser el lenguaje de programación o peculiaridades de la arquitectura.

Para documentar esta fase se generan varios modelos del sistema., representados por diagramas y descripciones textuales. Los modelos son de dos tipos: el modelo de procesos (representado por diagramas de flujo de datos) y el modelo de datos (representado por el MCD).

2.2. DIAGRAMA DE CONTEXTO

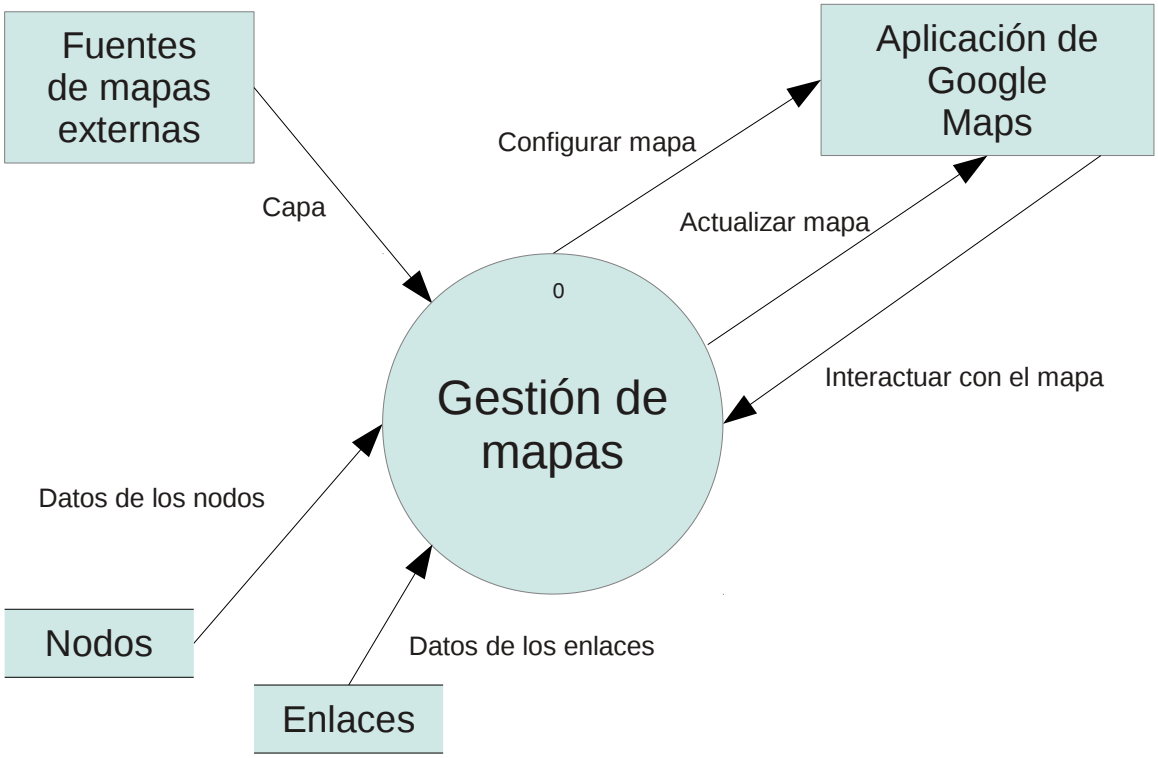


Ilustración 3: Diagrama de contexto de nuestra aplicación

En el diagrama de contexto podemos ver una vista por encima de lo que sería nuestro sistema una vez funcionando, teniendo en cuenta sólo las partes relevantes al proyecto. Las flechas representan movimiento de datos, los círculos representan procesos, los rectángulos entidades externas, y dos líneas paralelas horizontales los almacenes de datos.

En un diagrama de contexto sólo hay un proceso, el cual representa el sistema que vamos a desarrollar (nuestra solución). Lo que hay alrededor será pues, externo a nuestro sistema.

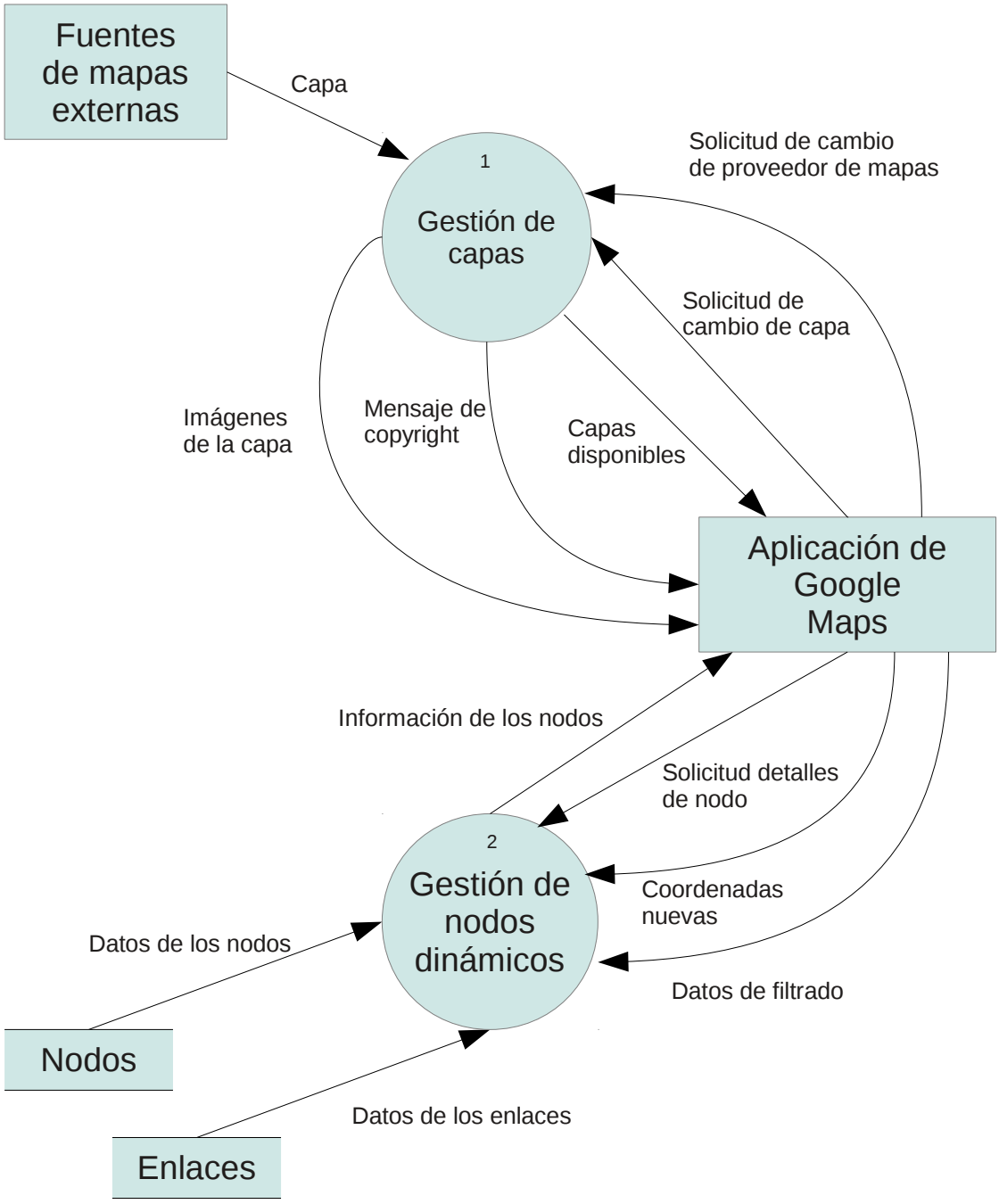
En este diagrama de contexto hemos considerado como entidades externas:

- La aplicación de Google Maps, puesto que es una cosa opaca que no se puede modificar excepto por su API. La aplicación nos pasa mensajes de eventos del usuario (pulsación de ratón, arrastre del mapa, menú seleccionado, etc.) y nosotros podemos responder a esos eventos solicitando cambios en el mapa o ignorándolos. Inicialmente podemos configurar el mapa dentro de unos límites.
- Las fuentes externas de mapas, como servidores WMS ó servidores de teselas como OpenStreetMap, Mapbox, etc. Su único propósito es responder a nuestras peticiones solicitando un mapa de una región concreta y devolver una serie de imágenes. A dicho conjunto de imágenes lo denominaremos “capa”, puesto que están relacionadas entre sí.
- Los almacenes de Guifi.net (Nodos y Enlaces), donde se guardan los datos de los nodos y los enlaces. Otra parte del sistema es la que se encarga de actualizar estos almacenes, y nosotros sólo accedemos para leer datos.

- **Nodos**.
  - **Descripción**. Guarda información sobre todos los nodos en el sistema.
  - **Estructura** (simplificada con sólo lo que nos interesa): Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo.
  - **Volumen medio**: 30.000.
- **Enlaces**.
  - **Descripción**. Guarda información sobre todos los enlaces en el sistema.
  - **Estructura** (simplificada con sólo lo que nos interesa: Código del enlace + Código del dispositivo enlazado + Código del nodo que tiene el dispositivo + Tipo del enlace + Estado del enlace.
  - **Volumen medio**: 40.000.

2.3. DFD. PRIMERA EXPLOSIÓN.

En la primera explosión se desmenuza el diagrama de contexto en varios subprocesos:



- **Gestión de capas**. Este proceso recibe peticiones para cambiar entre proveedores de mapas y pide a Google Maps que muestre al usuario las nuevas capas disponibles, luego selecciona la capa por defecto del nuevo proveedor, y muestra un mensaje de “copyright” de acuerdo con la capa seleccionada. Cuando se selecciona una capa nueva, nuestro proceso le envía a Google Maps las nuevas imágenes, y se actualiza el “copyright”.

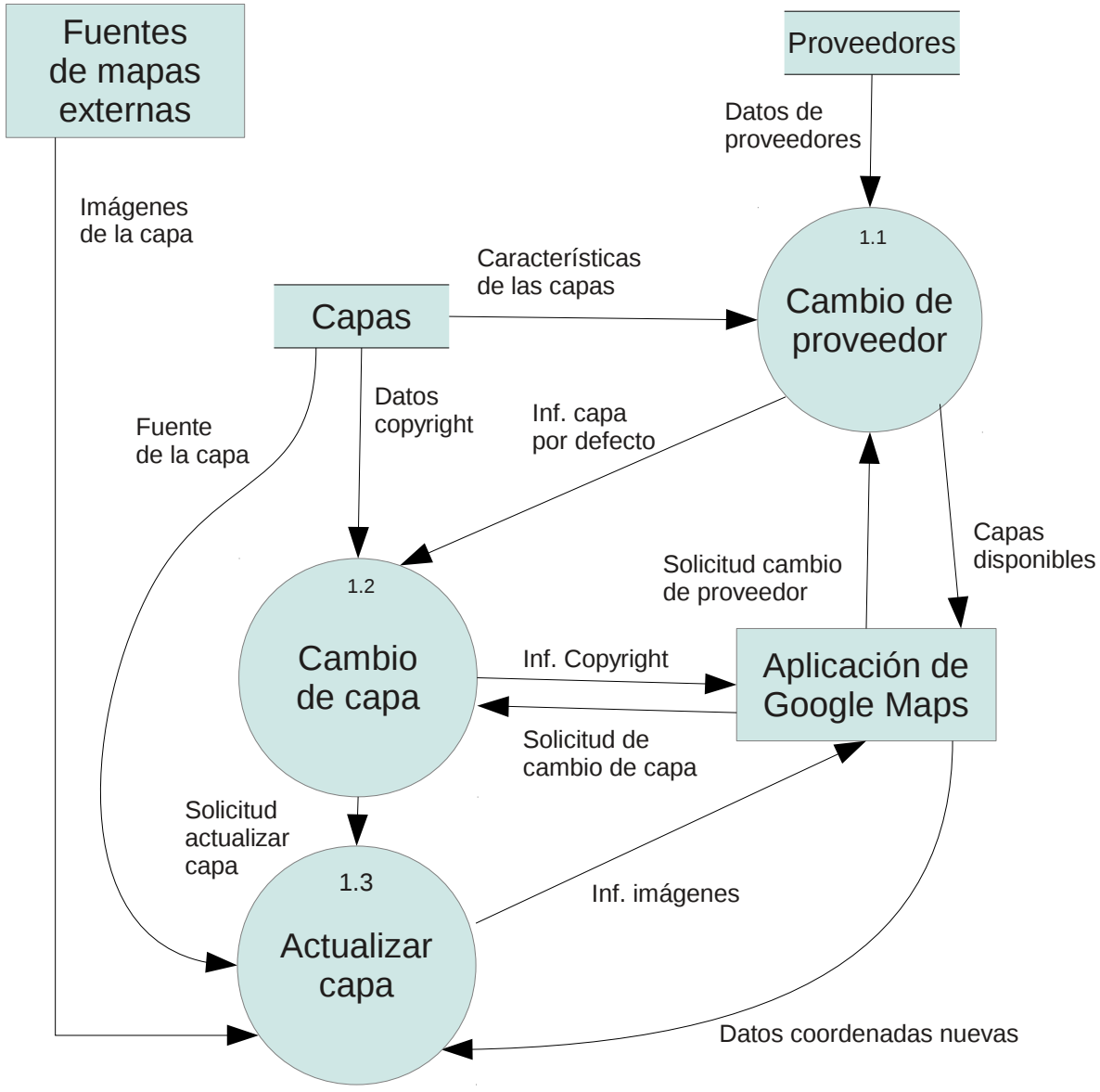
- **Gestión de nodos dinámicos**. Este proceso recibe peticiones de información sobre los nodos de una región, y en base a dos almacenes externos, se combinan los datos para proporcionar la información deseada. El proceso envía nueva información cuando las coordenadas cambian, o los parámetros de filtrado cambian. El proceso también envía información detallada de un nodo concreto cuando se solicita.

Los almacenes, al ser externos, se han descrito en el diagrama de contexto.

2.4. DFD. SEGUNDA EXPLOSIÓN.

La segunda explosión ahonda un poco más dentro de cada proceso de la primera explosión.

2.4.1. GESTIÓN DE CAPAS



2.4.1.1. ALMACENES

- **Proveedores**.
  - **Descripción**. Contiene la lista de proveedores de mapas disponibles. El almacén ya está lleno de proveedores y lo que haremos es sólo leerlo.
  - **Estructura**: Nombre de proveedor + Descripción del proveedor + 1{Nombre de capa}n.
  - **Volumen medio**: 3.
- **Capas**.
  - **Descripción**. Contiene la lista de capas disponibles. El almacén ya está lleno de capas y lo haremos es sólo leerlo.
  - **Estructura**: Nombre de capa + Descripción de la capa + Tipo de capa + Seleccionada por defecto + Copyright + 1{URL}n + Configuración.
  - **Volumen medio**: 10.

2.4.1.2. FLUJOS DE DATOS

Nombre	Origen	Destino	Descripción	Estructura
Datos de proveedores	Almacén Proveedores	Proceso Cambio de Proveedor	Datos de los proveedores que tenemos disponibles para seleccionar.	Nombre de proveedor + Descripción del proveedor + 1{Nombre de capa}n
Fuente de la capa	Almacén Capas	Proceso Actualizar Capa	Datos sobre cómo solicitar imágenes de una capa a una entidad externa.	1{URL}n + Configuración
Datos copyright	Almacén Capas	Proceso Cambio de Capa	Datos que se deben mostrar cuando una capa esté visible.	Copyright
Inf. copyright	Proceso Cambio de Capa	Entidad Aplicación de Google Maps	Información del copyright a mostrar en el mapa.	Texto del copyright formateado.
Características de las capas	Almacén Capas	Proceso Cambio de Proveedor	Datos de las capas	Nombre de capa + Descripción de la capa + Tipo de capa + Seleccionada por defecto
Capas disponibles	Proceso Cambio de Proveedor	Entidad Aplicación de Google Maps	Información de las capas disponibles de un proveedor determinado.	1{Nombre de capa + Descripción de la capa + Tipo de capa + Seleccionada por defecto}n
Solicitud cambio de proveedor	Entidad Aplicación de Google Maps	Proceso Cambio de Proveedor	Petición de cambiar a un proveedor diferente al actual.	Nombre del proveedor
Solicitud cambio de capa	Entidad Aplicación de Google Maps	Proceso Cambio de Capa	Petición de cambiar a una capa diferente a la actual.	Nombre de la capa

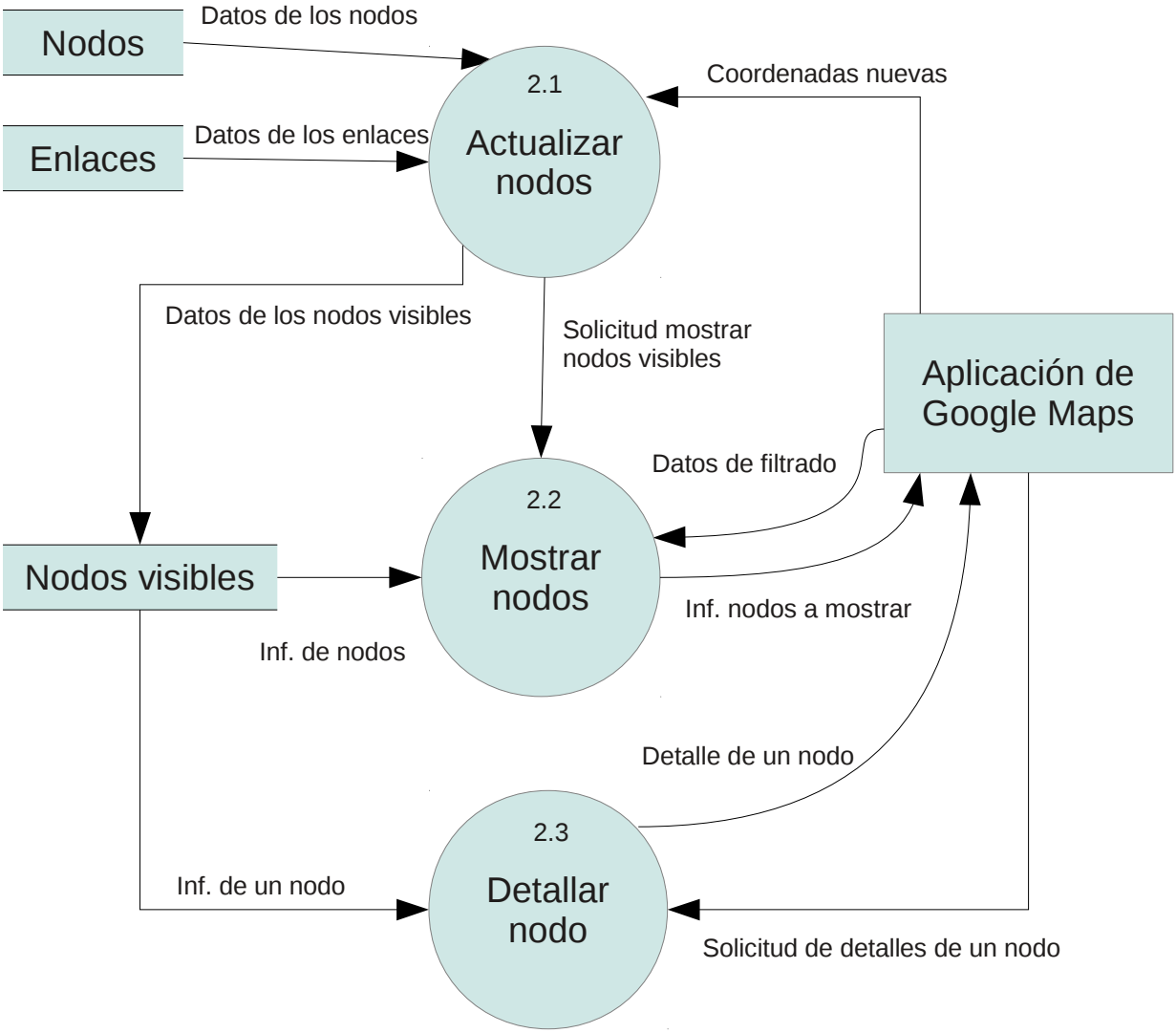
Nombre	Origen	Destino	Descripción	Estructura
Inf. capa por defecto	Proceso Cambio de Proveedor	Proceso Cambio de Capa	Petición de cambiar a la capa por defecto del proveedor actual.	Nombre de la capa
Imágenes de la capa	Entidad Fuentes de Mapas Externas	Proceso Actualizar Capa	Imagen (JPG, PNG, etc.).	<imagen>
Solicitud actualizar capa	Proceso Cambio de Capa	Proceso Actualizar Capa	Petición de actualizar la capa actual en respuesta a un cambio de la capa actual..	
Datos coordenadas nuevas	Entidad Aplicación de Google Maps	Proceso Actualizar Capa	Datos de las coordenadas nuevas del mapa y el nivel de zum.	Latitud + Longitud + Zum
Inf. imágenes	Proceso Actualizar Capa	Entidad Aplicación de Google Maps	Imagen (JPG, PNG, etc.).	<imagen>

2.4.1.3. PROCESOS

- **Cambio de proveedor**.
  - Leer información recibida de la Entidad Aplicación Google Maps (Nombre del proveedor a seleccionar).
  - Leer datos del proveedor.
  - Para las capas de ese proveedor
    - Leer datos de una capa
    - Si es la capa por defecto, llamar al Proceso Cambio de Capa con el nombre de la capa..
  - Devolver información sobre las capas disponibles para este proveedor a la Entidad Google Maps.
- **Cambio de capa**.
  - Leer información recibida del Proceso Cambio de Capa o de la Entidad Aplicación Google Maps (nombre de la capa a seleccionar).
  - Leer datos de copyright para esa capa del Almacén de Capas.
  - Formatear datos de copyright
  - Devolver datos de copyright a la Entidad Aplicación Google Maps.
  - Llamar al Proceso Actualizar Capa con el nombre de la capa a actualiar.
- **Actualizar capa**.
  - Leer información recibida del Proceso Cambio de Capa (Nombre de la capa a actualizar) o de la Entidad Aplicacíón Google Maps (Datos coordenadas nuevas).
  - Leer Fuente de la Capa (URLs y configuración de acceso) del Almacén de Capas.
  - Seleccionar una URL de la lista de URLs (en secuencia).

- Solicitar imágenes a la Entidad Fuentes de Mapas Externas con los datos de la URL, la configuración de acceso y las coordenadas.

2.4.2. GESTIÓN DE NODOS DINÁMICOS



2.4.2.1. ALMACENES

Los almacenes de Nodos y Enlaces fueron descritos en el Diagrama de Contexto.

- **Nodos visibles**.
  - **Descripción**. Contiene información sobre los nodos que están dentro de la región que abarca el mapa. La información contenida aquí es una mezcla de datos de los almacenes de Nodos y Enlaces, y entre otras cosas sirve como caché.
  - **Estructura**. Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo + Número de enlaces.

- **Volumen medio**: 300.

2.4.2.2. FLUJOS DE DATOS

Nombre	Origen	Destino	Descripción	Estructura
Datos de los nodos	Almacén de Nodos	Proceso Actualizar Nodos	Datos de los nodos dentro de una región.	Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo
Datos de los enlaces	Almacén de Enlaces	Proceso Actualizar Nodos	Datos de los enlaces cuyo código de nodo está dentro de una región.	Código del nodo que tiene el dispositivo + Tipo del enlace + Estado del enlace
Coordenadas nuevas	Entidad Aplicación de Google Maps	Proceso Actualizar Nodos	Datos de las nuevas coordenadas del mapa.	Latitud mínima + Longitud mínima + Latitud máxima + Longitud máxima
Datos de los nodos visibles	Proceso Actualizar Nodos	Almacén Nodos Visibles	Datos combinados de los nodos dentro una región y sus enlaces.	Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo + Número de enlaces
Inf. de nodos	Almacén Nodos Visibles	Proceso Mostrar Nodos	Información de los nodos dentro de una región y cuántos enlaces tiene cada uno.	Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo + Número de enlaces
Solicitud mostrar nodos visibles	Proceso Actualizar Nodos	Proceso Mostrar Nodos	Petición de mostrar los nodos del almacén de nodos visibles con los filtros actuales	
Datos de filtrado	Entidad Aplicación de Google Maps	Proceso Mostrar Nodos	Datos de qué nodos no deben verse	(Estado del nodo) + (Número de enlaces)
Inf. nodos a mostrar	Proceso Mostrar Nodos	Entidad Aplicación de Google Maps	Información de qué nodos se mostrarán	1{Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Número de enlaces}n
Inf. de un nodo	Almacén Nodos Visibles	Proceso Detallar Nodo	Información detallada de un solo nodo	Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo + Número de enlaces
Detalle de un nodo	Proceso Detallar Nodo	Entidad Aplicación de Google Maps	Información detallada y formateada de un solo nodo	Código del nodo + Nombre del nodo + Latitud + Longitud + Estado del nodo + Estabilidad del nodo + Número de enlaces

Nombre	Origen	Destino	Descripción	Estructura
Solicitud de detalles de un nodo	Entidad Aplicación de Google Maps	Proceso Detallar Nodo	Petición para consultar información detallada sobre un nodo concreto	

2.4.2.3. PROCESOS

- **Actualizar nodos**.
  - Leer información de la entidad Aplicación de Google Maps (Coordenadas nuevas).
  - Solicitar al almacen de Nodos aquellos nodos que están dentro de las coordenadas.
  - Solicitar al almacen de Enlaces el número de enlaces que están relacionados con los. nodos anteriores
  - Borrar almacén de nodos visibles.
  - Almacenar los datos de los nodos + número de enlaces en el Almacén de Nodos Visibles.
  - Llamar al Proceso Mostrar Nodos para mostrar los nodos del Almacén de Nodos Visibles.
- **Mostrar nodos**.
  - Leer información de la Entidad Aplicación de Google Maps (Datos de filtrado)
  - Leer información del Almacén Nodos Visibles
  - Descartar los nodos que no coincidan con el filtro
  - Enviar la información a la Entidad Aplicación de Google Maps.
- **Detallar nodo**.
  - Leer información de la Entidad Aplicación de Google Maps (código de nodo).
  - Leer información del Almacén Nodos Visibles para el nodo solicitado.
  - Formatear la información del nodo
  - Enviar el resultado a la Entidad Aplicación de Google Maps

2.5. MODELO CONCEPTUAL DE DATOS

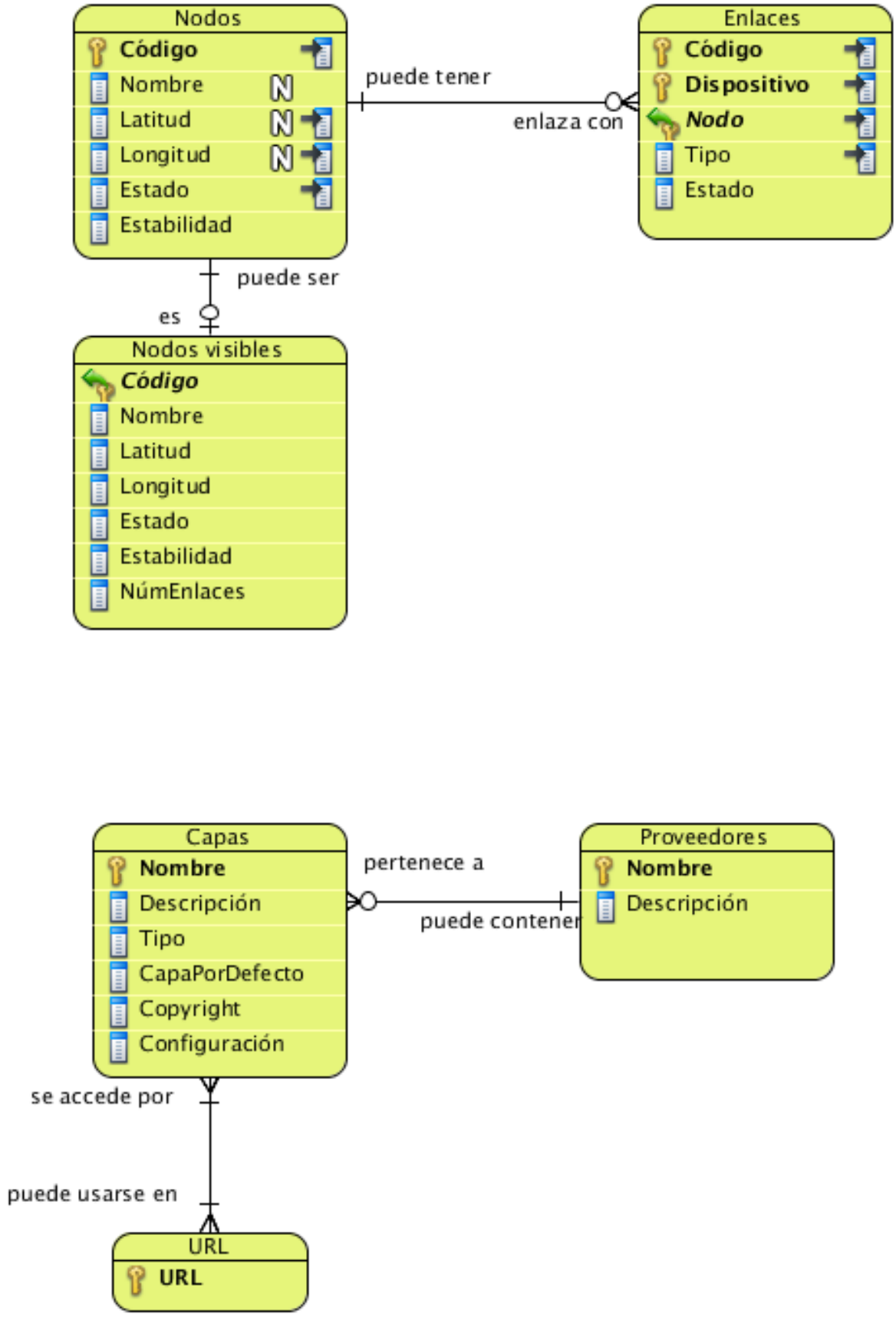


Ilustración 4: Gráfico del Modelo Conceptual de Datos

2.5.1. RELACIONES

- **Nodos/Enlaces** (puede tener/enlaza con): Un nodo puede tener cero enlaces (cuando se crea) y no tiene límite en el número de enlaces que puede tener (no obligatoreidad). Un enlace, sin embargo, sólo puede pertenecer a un nodo (obligatoreidad). Hay que notar que lo que se pinta en un mapa como una línea de nodo a nodo, se modela conceptualmente como dos enlaces con el mismo código, pero distinto dispositivo. El enlace puede ser dentro del nodo (enlaces de cable) entre dos dispositivos. Esta relación estaba ya dentro del Sistema Inicial.
- **Nodos/Nodos visibles** (puede ser/es): Un nodo puede ser visible (no obligatoreidad). Un nodo visible es un nodo (obligatoreidad).
- **Capas/Proveedores** (pertenece a/puede contener): Una capa pertenece a un solo proveedor (obligatoreidad). Un proveedor puede contener un número indeterminado de capas o ninguna (no obligatoreidad).
- **Capas/URL** (se accede por/puede usarse en): Para obtener una capa se puede acceder por varias URL, pero al menos tiene que haber una (obligatoreidad). Una misma URL puede usarse para varias capas, pero al menos debe usarse en alguna (obligatoreidad).

2.5.2. ENTIDADES

Nodo			
Descripción: Representa a un nodo o ubicación dentro de la red.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
Código	Clave primaria	Código que identifica cada nodo	Entero (9)
Nombre		Nombre del nodo	Cadena (40)
Latitud		Cercanía al ecuador	Real (10,6)
Longitud		Cercanía al meridiano	Real (10,6)
Estado		Estado de funcionamiento del nodo (Operativo, Proyectado, etc).	Cadena (40)
Estabilidad		Cadena que indica si el nodo es fijo (funciona las 24 horas) o no (“Yes”, “No”)	Cadena (25)

Enlace			
Descripción: Representa un extremo de un enlace entre dos dispositivos.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
Código	Clave primaria	Código del enlace (entendido como la unión entre dos dispositivos)	Entero (9)
Dispositivo	Clave primaria	Dispositivo	Entero (9)
Nodo	Clave ajena	Código del nodo donde está el dispositivo	Entero (9)
Tipo		Tipo del enlace (“WDS”, “cliente”, etc)	Cadena (40)
Estado		Estado de funcionamiento del enlace (Operativo, Proyectado, etc).	Cadena (40)

Nodo visible			
Descripción: Representa a un nodo que está dentro de la región actual.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
Código	Clave primaria	Código que identifica cada nodo	Entero (9)
Nombre		Nombre del nodo	Cadena (40)
Latitud		Cercanía al ecuador	Real (10,6)
Longitud		Cercanía al meridiano	Real (10,6)
Estado		Estado de funcionamiento del nodo (Operativo, Proyectado, etc).	Cadena (40)
NúmEnlaces		Número de enlaces que tiene este nodo	Entero (9)

Capa			
Descripción: Representa a una capa del mapa, independientemente del contenido.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
Nombre	Clave primaria	Nombre de la capa	Cadena (40)
Descripción		Descripción de la capa	Cadena (40)
Tipo		Tipo de la capa (capa base, o capa con transparencia)	Boolean
CapaPorDefecto		Cualidad de capa por defecto del proveedor (sí, no)	Boolean
Copyright		Información de Copyright de la capa	Cadena (40)
Configuración		Otros parámetros de configuración para poder acceder a la capa	Cadena (40)

Proveedor			
Descripción: Representa a un proveedor de mapas (Google, OSM, etc). Su única misión es contener los datos de las capas.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
Nombre	Clave primaria	Nombre del proveedor	Cadena (40)
Descripción		Descripción del proveedor	Cadena (40)

URL			
Descripción: Representa una URL para acceder a un servicio web de mapas. Es típico que se pueda acceder a una misma capa por varias URL (con subdominios) para acelerar las peticiones.			
Atributos			
Nombre	**Tipo**	**Descripción**	**Dominio**
URL	Clave primaria	Contenido de la URL	Cadena (40)

CAPÍTULO 3: DISEÑO

ÍNDICE

CAPÍTULO 3: DISEÑO 31
3.1. INTRODUCCIÓN 31
3.2. DISEÑO DE INTERFACES 32
3.2.1. PANEL LATERAL 33
3.2.2. SELECTOR DE CAPAS 36
3.2.3. CAPA DE NODOS DINÁMICOS 40
3.2.4. CAPAS EXTRA 43
3.3. DISEÑO DE LA BASE DE DATOS 49
3.4. MODELO DE DISEÑO 49

3.1. INTRODUCCIÓN

En la fase anterior se generaron unos modelos conceptuales del sistema. En esta fase a esos modelos se les aplica restricciones de implementación, como puede ser la arquitectura elegida o el lenguaje de programación.

3.2. DISEÑO DE INTERFACES

Comenzaremos con el diseño de interfaces, pues éste da pie a definir el resto del diseño. Las interfaces dan una buena idea de los casos de uso reales de la aplicación y ayudan a hacerse una idea rápida y más visual de lo que se pretende conseguir.

Siguiendo con uno de los objetivos del sistema, que es elaborar una interfaz amigable no muy diferente de la existente, todos los elementos de la interfaz se han intentado asemejar a los ya proporcionados por la aplicación de Google Maps.

Además, puesto que la recomendación futura para Guifi es portar el código de Google Maps a OpenLayers, para aquellos elementos de la interfaz no presentes en Google Maps, pero sí presentes en OpenLayers, se ha elaborado un estilo nuevo, mezcla de ambos. Se ha usado una paleta con tonos de grises cercanos al blanco, por ser ésta la paleta de Google Maps, y se ha intentado imitar la elegancia.

Ilustración 5: Interfaz del sistema inicial

En la Ilustración 5 podemos ver el aspecto de la interfaz del sistema inicial (antes de implementar las mejoras).

En la parte izquierda podemos ver dos controles estándar de Google Maps: el panel de desplazamiento y la barra de zum ("*pan and zoom*"). En la parte inferior esta el logotipo de Google y la información del copyright. En el centro hay un rectángulo que delimita la zona de la red que estamos explorando.

En la parte superior derecha está el menú estándar para cambiar de capas (Mapa / Satélite) y un botón para activar o desactivar la capa estática de Guifi.net.

3.2.1. PANEL LATERAL

En Google Maps (ver Ilustración 5) las capas se seleccionan con un menú superior derecho con dos botones blancos (Mapa / Satélite) los cuales sirven para elegir capas base. Al mantener el puntero del ratón por encima de ellos aparece una lista de capas superpuestas que se pueden seleccionar (Relieve / Etiquetas, ver Ilustración 9).

Queremos que los antiguos usuarios no vean cambios sustanciales en la forma de trabajar que tenían antes, es decir, que deben de poder cambiar entre capas como lo han hecho siempre, usando un menú superior derecho con varios botones.

Nosotros necesitamos, además de poder cambiar entre capas, poder cambiar entre proveedores de mapas, seleccionar filtros, etc.

Las funciones nuevas, que podríamos decir que son “avanzadas”, tenemos que ponerlas en algún otro sitio que no sea el menú, a poder ser que no estorbe mucho, pero que se pueda descubrir.

OpenLayers nos ofrece en su librería un panel lateral derecho para seleccionar capas base y capas superpuestas (ver Ilustración 8). Un panel similar parece la elección indicada para colocar todas estas funciones nuevas:

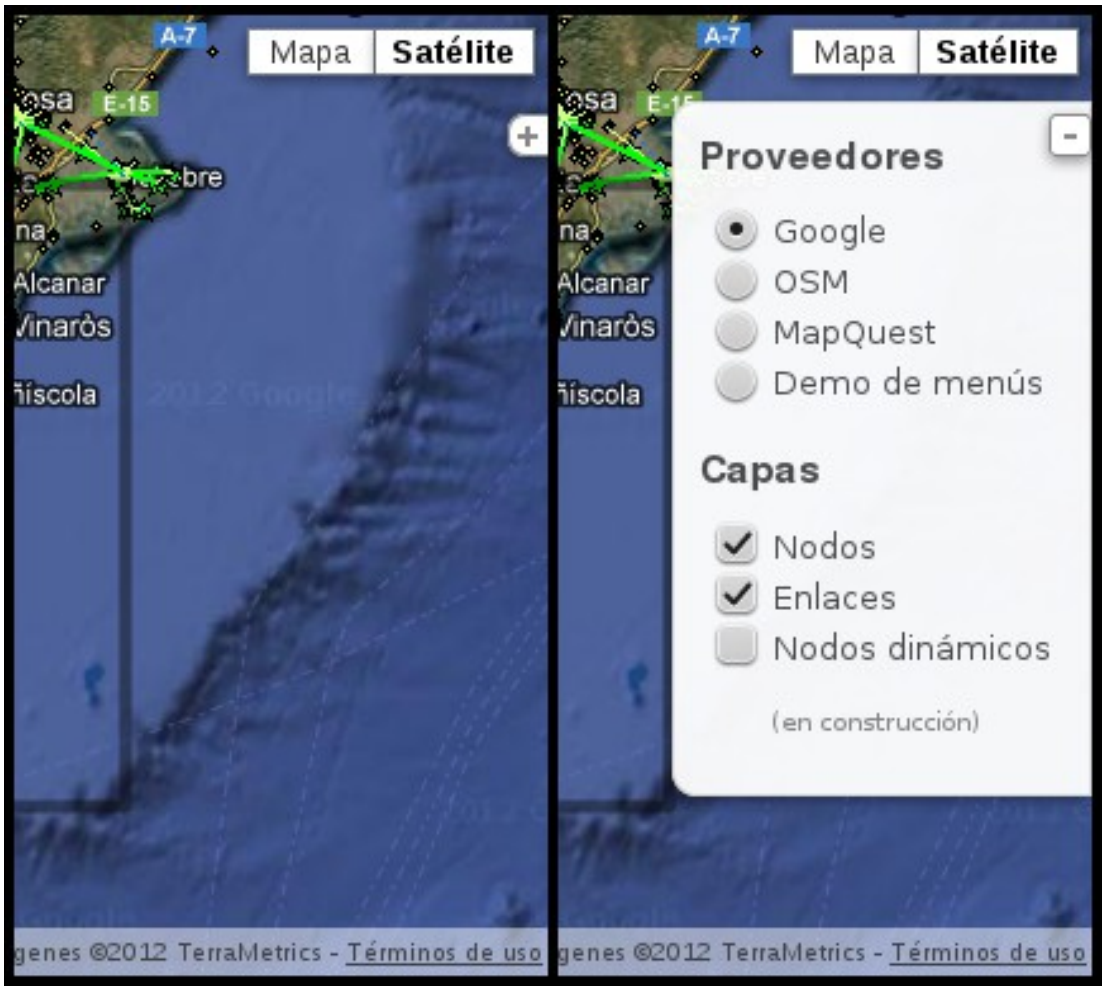


Ilustración 6: Propuesta de panel lateral para cambiar entre los proveedores y realizar filtros sobre los nodos

Este panel aparece inicialmente como un pequeño botón blanco con el símbolo “+”. Al pulsar en él, se despliega un panel donde podemos encontrar todas las funciones avanzadas. El botón “+”

pasa a mostrar ahora “-”. Pulsando sobre él otra vez el panel se oculta de nuevo, y el botón vuelve a mostrar “+”.

En el panel tenemos dos secciones, la superior para cambiar entre proveedores, y la inferior para realizar operaciones adicionales con capas superpuestas, como activar / desactivar la capa de Guifi (nodos o enlaces), mostrar / ocultar nodos dinámicos, filtrar por tipo de nodo, etc.

Al seleccionar un proveedor diferente, la capa del fondo cambia a la capa por defecto de ese proveedor.

Adicionalmente, dependiendo de las capas disponibles que tenga ese proveedor, el menú superior derecho cambia para mostrar las capas disponibles (ver sección 3.2.2 Selector de capas).

Además, al cargar las capas de Guifi, se muestra un icono animado que representa que la capa está todavía cargando. Cuando termina la carga el icono desaparece:

Ilustración 7: Detalle del señalizador de capa cargando (icono animado

El panel se ha diseñado para ser parecido al panel lateral de OpenLayers, pero con la paleta de colores y el minimalismo de Google.

Esto facilitará una migración futura, puesto que los usuarios ya se habrán acostumbrado a tener un panel lateral.

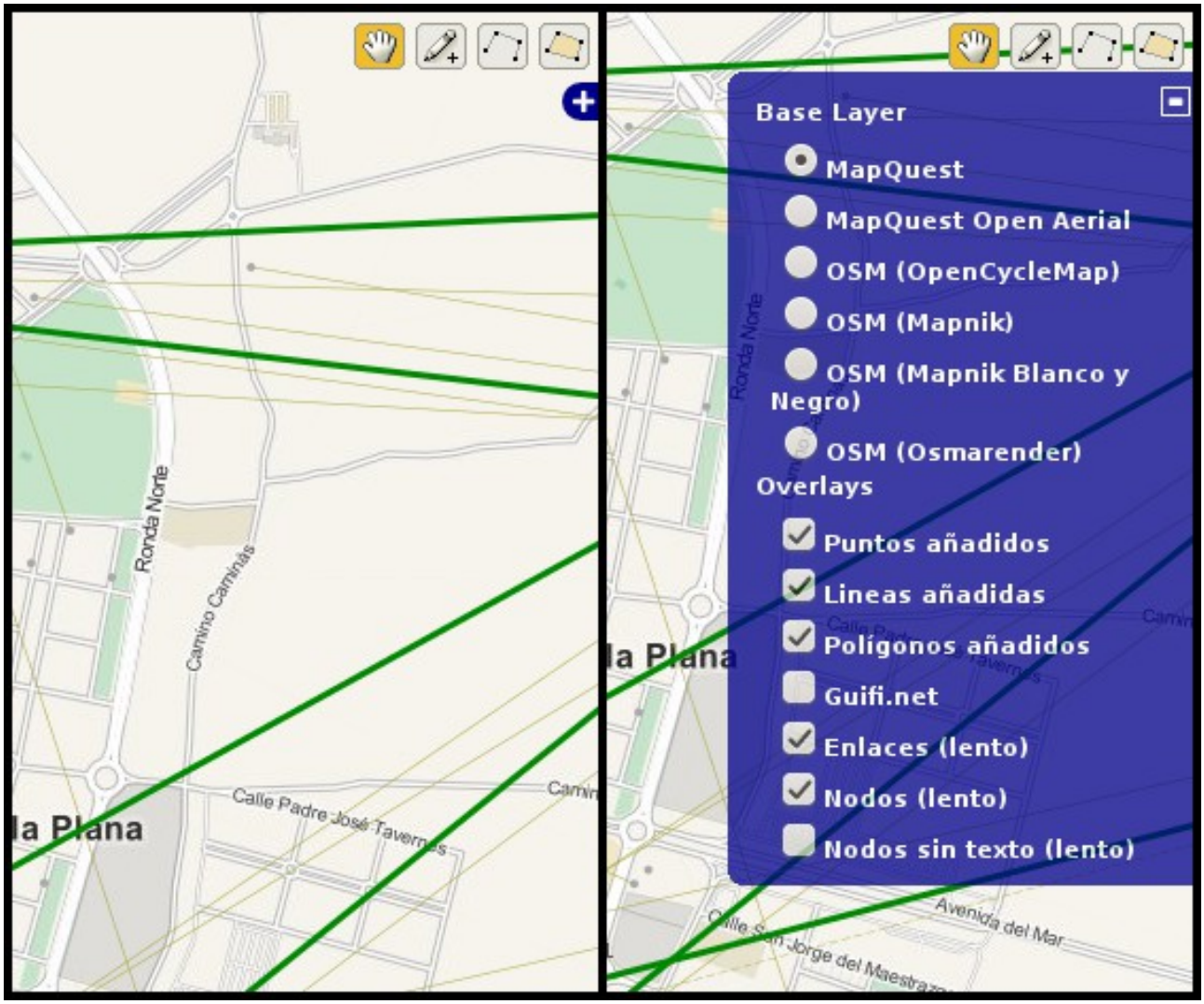


Ilustración 8: Panel lateral para cambiar entre capas que incluye por defecto la librería OpenLayers (con varias capas añadidas)

3.2.2. SELECTOR DE CAPAS

El selector de capas que tiene Google por defecto es un selector minimalista con dos botones blancos: uno para seleccionar mapas dibujados y otro para seleccionar fotografías reales por satélite.

Adicionalmente, al pasar el puntero del ratón por encima, podemos seleccionar varias capas superpuestas, como curvas de nivel para el mapa dibujado, o carreteras y nombres de calles y ciudades para las fotos por satélite.

Ilustración 9: Detalle del selector de capas original de Google

Podemos entender entonces, que los botones iniciales muestran una "Categoría" de mapas (dibujo o fotos reales), y las opciones que hay debajo de cada uno de ellos son solamente mapas adicionales que entran dentro de esa categoría.

Puesto que el selector de Google no se puede personalizar más allá de añadir capas propias de Google y poco más, lo que haremos será construir un nuevo elemento de la interfaz que usaremos para seleccionar nuevas capas.

Este nuevo elemento será un control personalizado con una apariencia parecida al selector de Google. Cuando tengamos seleccionado un proveedor distinto a Google, se mostrará nuestro selector personalizado, y en caso contrario, se mostrará el de Google.

Ilustración 10: Nuevo selector de capas seleccionando una capa de la categoría "Satélite".

Puesto que no podemos prever qué uso le daremos al control en un futuro, qué capas nuevas

añadiremos, etc. el control tiene que ser lo más extensible posible.

Para ello se ha decidido permitir un número ilimitado de botones o “Categorías” en la parte superior.

Al dejar el puntero del ratón un tiempo sobre la categoría activa, se abrirá un menú desplegable como el de una aplicación normal.

Los componentes de dicho menú desplegable serán:

- **Secciones**. Cada sección constará de:
  - **Título de la sección** (opcional). Al dejar el puntero encima saldrá un mensaje de ayuda (“tooltip”).
  - **Una hilera de casillas seleccionables o una hilera de botones de opción**. A la derecha de cada botón se mostrará una etiqueta. Al pasar el puntero por encima de la etiqueta o del botón el contenido se sombreará, y si esperamos más tiempo saldrá un mensaje de ayuda (“tooltip”). A su vez, cada casilla/botón podrá contener:
    - **Una hilera de casillas seleccionables o una hilera de botones de opción**. El anidado de botones se podrá alargar hasta donde se quiera.
  - **HTML extra** (opcional). La posibilidad de que podamos añadir directamente HTML puede servir para extender la funcionalidad del control. Podemos añadir por ejemplo texto explicativo, imágenes, o algún elemento interactivo extra.
- **Separadores**. Se colocarán automáticamente para separar secciones.

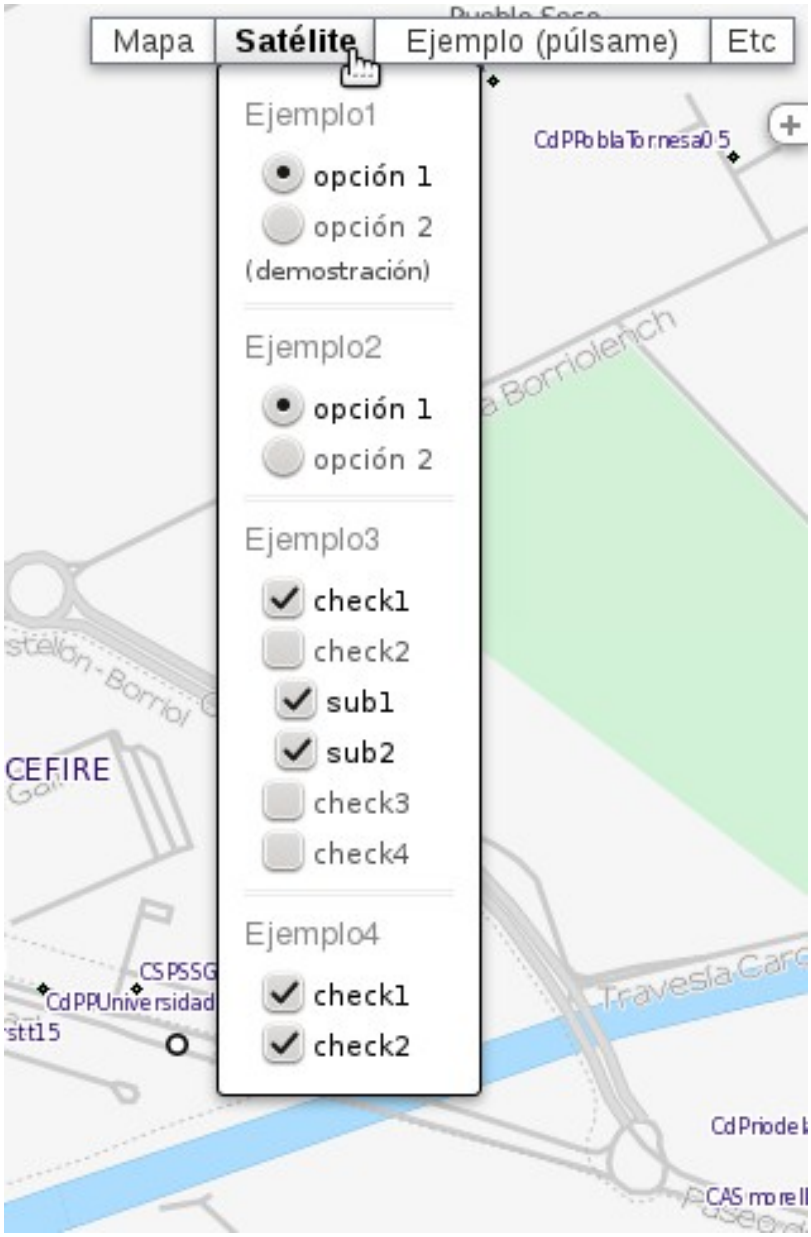


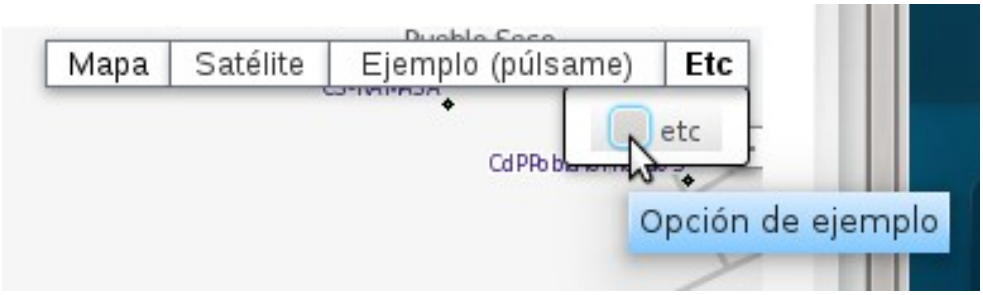
Ilustración 11: Muestra de la personalización posible en el selector de capas

Al seleccionar una opción con subelementos, se seleccionarán también los subelementos. Al mover el puntero del ratón fuera de los límites del control, el menú desplegable se replegará.

Podemos usar la funcionalidad de este control para activar o desactivar capas, subcapas, o para realizar cualquier otra acción sobre el mapa, como mostrar una leyenda, etc. Para ello el control tendrá que ser capaz de enviar eventos cuando una opción se selecciona.

Ilustración 12: Ejemplo de posible funcionalidad para el selector de capas

Además de toda esa capacidad de personalización, debe permitir crear menús minimalistas parecidos a los que tiene Google por defecto, que consten de una única opción sin demasiada funcionalidad (como el menú para activar las etiquetas en la capa de satélite):



Esta funcionalidad permitirá diseñar menús para usuarios comunes y usuarios avanzados al mismo tiempo.

3.2.3. CAPA DE NODOS DINÁMICOS

Para dibujar nodos de forma dinámica tenemos que añadir una nueva capa transparente al mapa, y dibujar los nodos en ella según las características del nodo y los filtros en uso.

Puesto que al recuperar los nodos tenemos ya información básica sobre los mismos, como el tipo de nodo (cliente o supernodo), estado (operativo, proyectado, etc), nombre, número de enlaces, etc. podemos personalizar cada nodo con un estilo diferente según cada una de estas características.

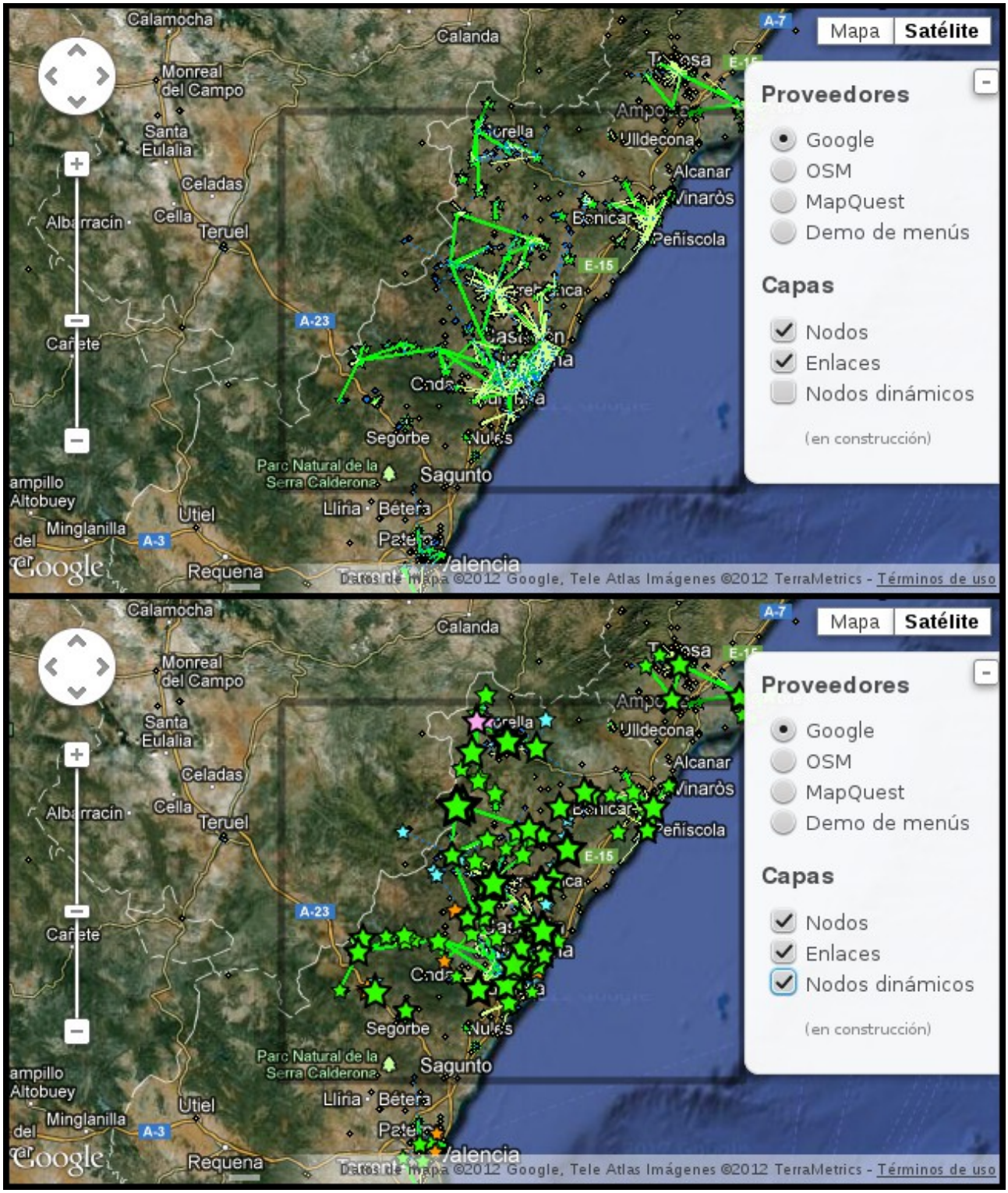


Ilustración 13: Comparación entre nodos dinámicos visibles u ocultos

La capa de nodos dinámicos se activará u ocultara al pulsar una opción en el panel lateral. Esto es así porque hemos dicho que los usuarios antiguos deben poder ver la interfaz que siempre han usado por defecto.

Los distintos tipos de nodo se dibujarán de la siguiente manera:

- **Nodos cliente**: como un punto.
- **Supernodos**: como una estrella. La forma de la estrella se ha elegido por ser también el símbolo que tienen los supernodos en la capa estática de nodos. Otra forma que se investigó fue el círculo. El tamaño del símbolo dependerá del número de enlaces que tenga el supernodo, destacando así la "importancia" que tiene ese supernodo. Puesto que hay nodos que tienen hasta 500 enlaces, la fórmula que usaremos para calcular el tamaño del nodo no será lineal, sino la raíz n-sima del número de enlaces. Adicionalmente el grosor del borde también dependerá del número de enlaces.
- **Estados**. Para los estados se usarán los mismos colores que se usan en la web de Guifi, con algunas excepciones.
  - Proyectado: Color azul claro.
  - Reservado: Color rosa. Este color difiere del de la web, blanco, porque se podría confundir con los controles del mapa, y por compartir el color con otro estado (borrado).
  - En construcción: Color amarillo claro.
  - En pruebas: Color naranja oscuro.
  - Operativo: Color verde.
  - Borrado: Color negro. Este color difiere del de la web por el mismo motivo que el del estado reservado. Se eligió el negro por ofrecer una idea más intuitiva de un nodo borrado (el negro sugiere inactividad).
- **Estabilidad**. Cuando un nodo es estable significa que la intención es que permanezca encendido constantemente, mientras que cuando no lo es significa que puede estar inactivo a temporadas. En el caso de que no sea estable, el color del nodo se "desatura", es decir, se vuelve más grisáceo respecto a su color original.

Al pasar el puntero del ratón por encima de un nodo, se muestra su nombre:

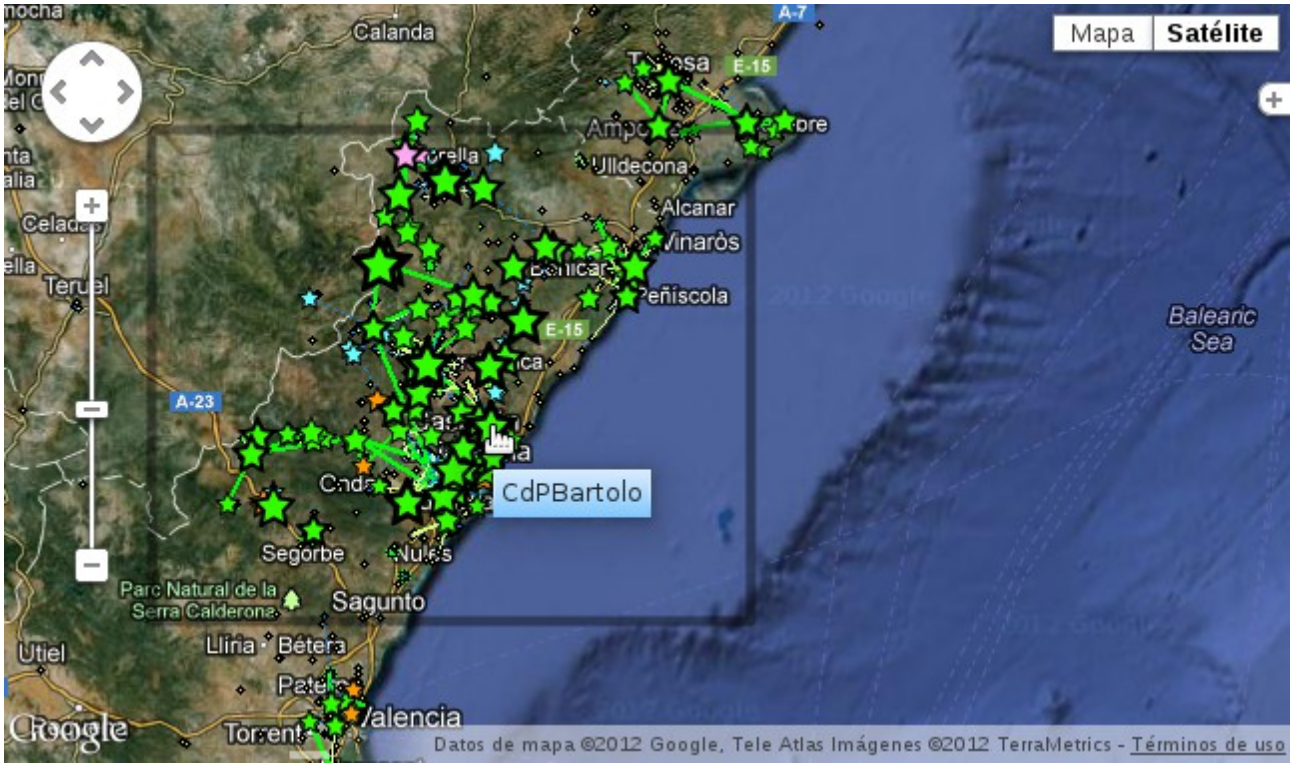


Ilustración 14: Detalle del mensaje emergente ("tooltip") que aparece al pasar el puntero del ratón por encima de un nodo

Finalmente, al pulsar sobre el nodo, aparece un globo informativo con la siguiente información:

- **Nombre del nodo**
- **Enlace de la página de gestión del nodo** (sobre el nombre)
- **Clasificación del nodo** (supernodo o no)
- **Número de enlaces inalámbricos**
- **Estado del nodo** (Operativo, etc). El texto del estado aparece del color correspondiente al estado, como vimos antes. En algunos casos se han cambiado ligeramente los colores para aumentar el contraste sobre fondo blanco.
- **Estabilidad del nodo**

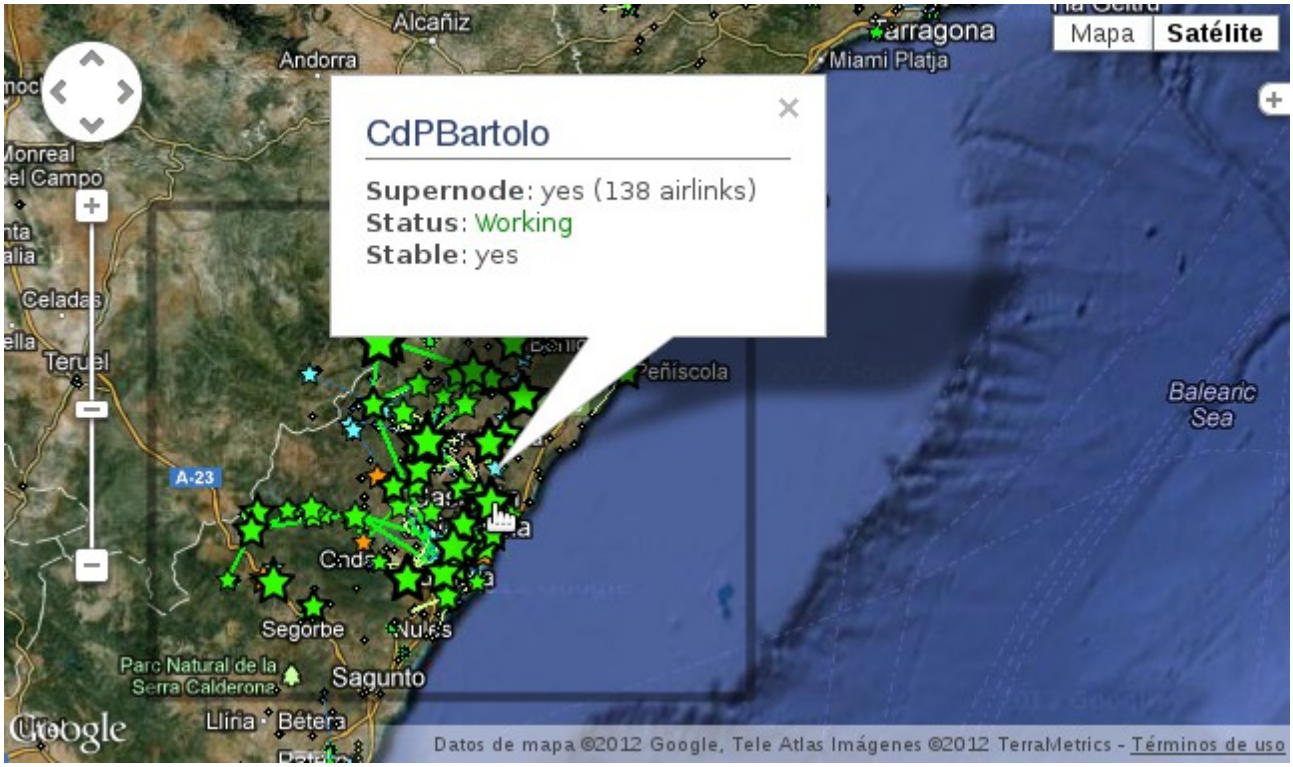


Ilustración 15: Detalle del globo informativo que aparece al pulsar sobre un nodo

3.2.4. CAPAS EXTRA

Existen muchos servicios de mapas gratuitos y/o con datos libres (basados en datos de OpenStreetMap).

A continuación se muestra una serie de ilustraciones con aquellos que parecen ser más útiles para la aplicación de Guifi.net.

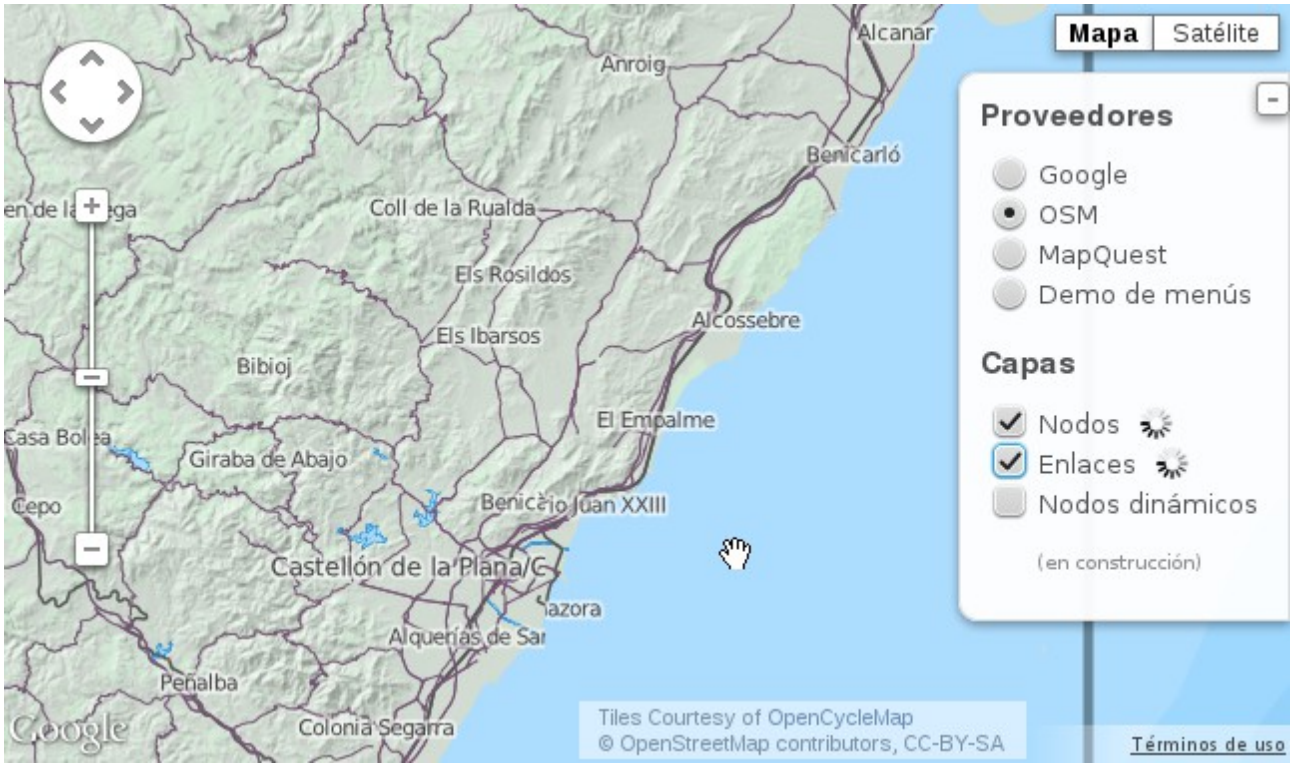


Ilustración 16: Capa OpenCycleMap "Landscape". Esta capa está diseñada para acentuar los elementos naturales. Contiene relieve sombreado ("shaded relief").

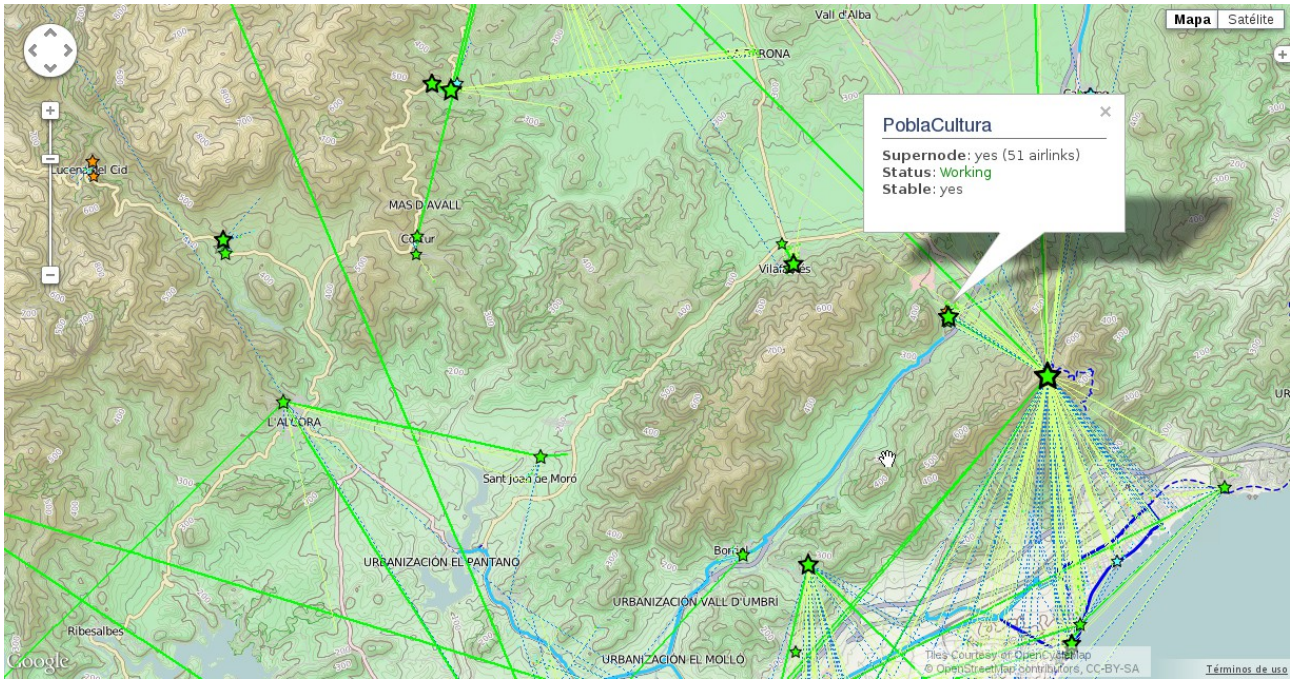


Ilustración 17: Capa OpenCyclemap por defecto. Esta capa está diseñada para acentuar elementos útiles para ciclistas. Contiene curvas de nivel ("contours"). Esto último hace que sea ideal para planificar enlaces cerca de zonas montañosas, como se puede ver.

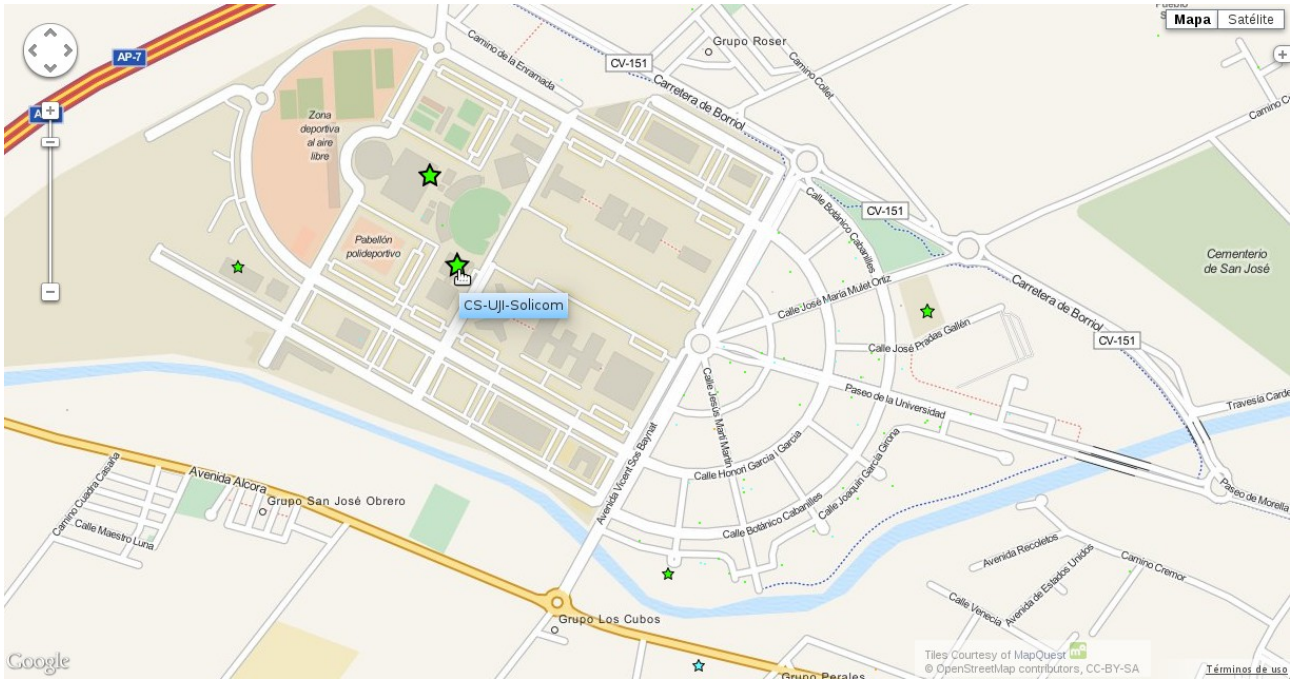


Ilustración 18: Capa Mapquest por defecto. Esta capa hace uso de tonos pastel y líneas suaves, sin mucho contraste, de forma que no molesta a la hora de superponer información encima. Compárese a la siguiente capa en cuanto a complejidad.

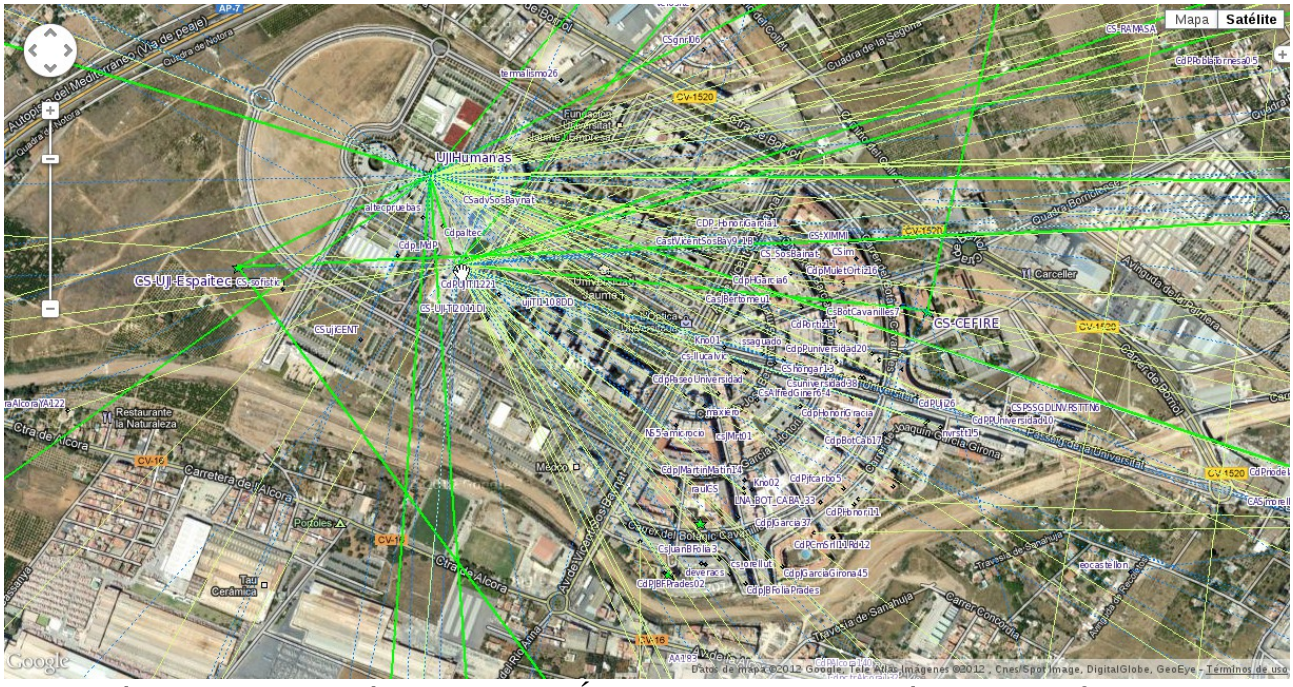


Ilustración 19: Capa de Satélite de Google. Ésta es la vista que ya teníamos en Guifi.net por defecto. A diferencia de la capa anterior, esta capa está llena de elementos con mucho contraste, que dificultan la visualización de mucha información a la vez. Como ventaja se puede señalar que permite localizar mejor las azoteas de los edificios que otras capas.

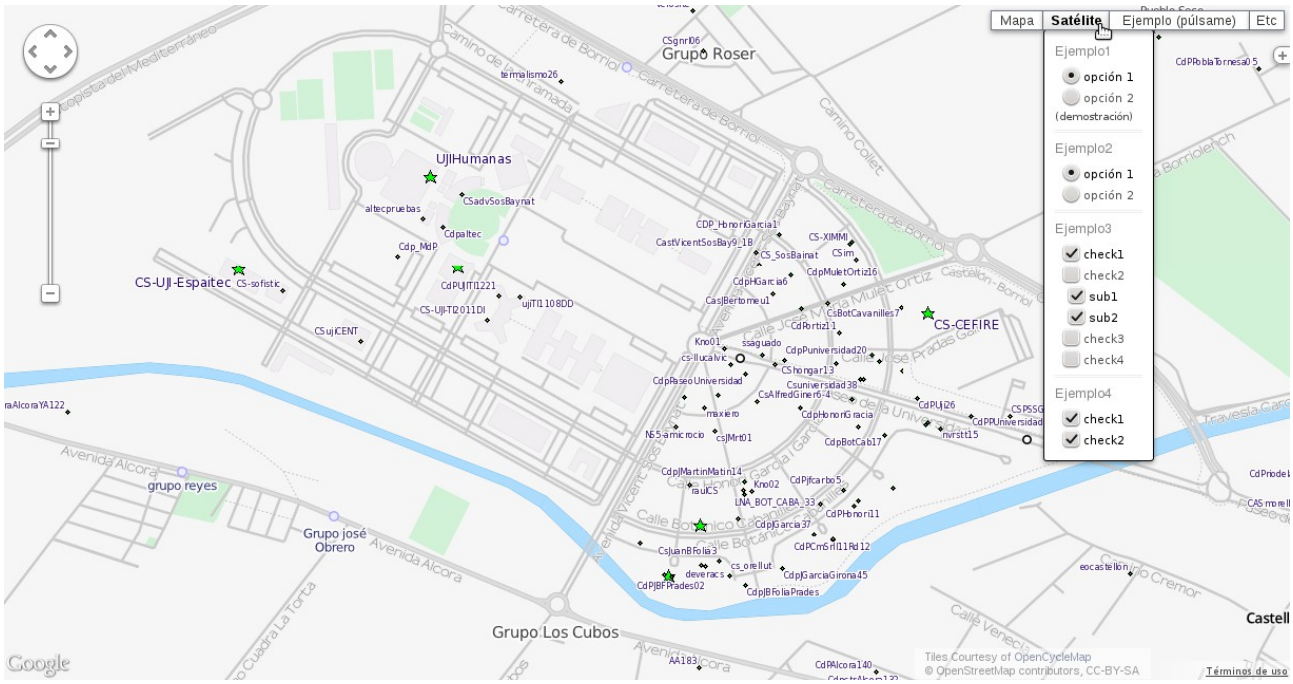


Ilustración 20: Capa OpenCycleMap "Transport". Esta capa está diseñada para realzar los elementos de transporte urbano, como carreteras, líneas de autobús, etc. Hace uso de tonos muy suaves en los que abunda el gris, con lo que es apta para impresión, y ofrece una vista de la ciudad más orientada a conexiones entre zonas que a edificios o lugares de interés.

Ilustración 21: Capa de Satélite de Google con etiquetas (vista global). Cuando tenemos muy reducido el nivel de zum, una capa con mucho contraste puede dar una ide más general del estado de la red, ya que los datos se muestran más como masas sólidas que como puntos y líneas.

Ilustración 22: Capa de Satélite de Google (con etiquetas) + vista de nodos dinámicos. En bajos niveles de zum la capa de nodos dinámicos proporciona una vista más clara de los nodos más importantes de la red, que destacan por su tamaño.

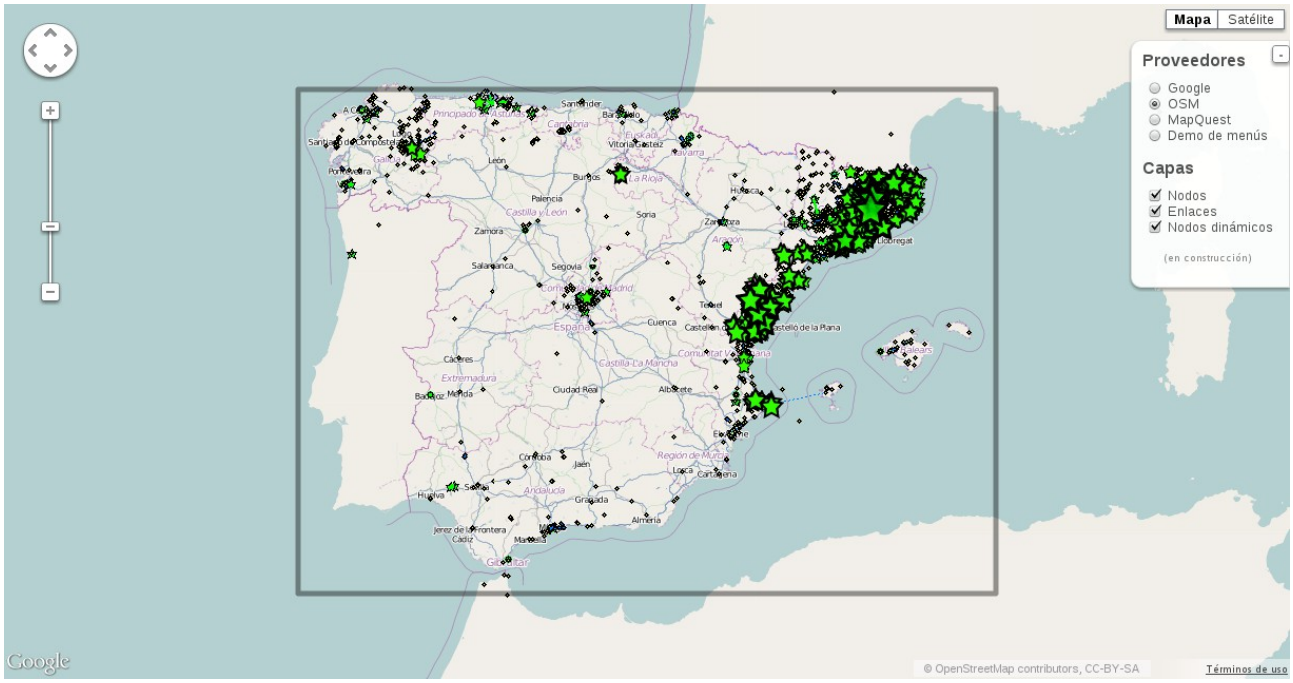


Ilustración 23: Capa de OpenStreetMap por defecto. Hay un ejemplo del montaje de un servidor de teselas con esta capa en el Capítulo 5: DESPLIEGUE. A bajos niveles de zum esta capa proporciona una vista clara del estado de la red. Los colores cercanos al blanco proporcionan también un alto contraste respecto a la información de los nodos. El bajo contraste la hace apta para impresión (ahorro de tinta).

Ilustración 24: Capa OpenAerial. Esta es una capa de fotografías por satélite con datos de carácter libre, donados por cortesía de la NASA. Estas fotografías están incluídas en el ya difunto proyecto OpenAerial para la elaboración de una base de datos de ortofotografía aérea libre.

3.3. DISEÑO DE LA BASE DE DATOS

Puesto que algunas entidades ya están presentes en la base de datos de Guifi.net, no es necesario volver a diseñarlas.

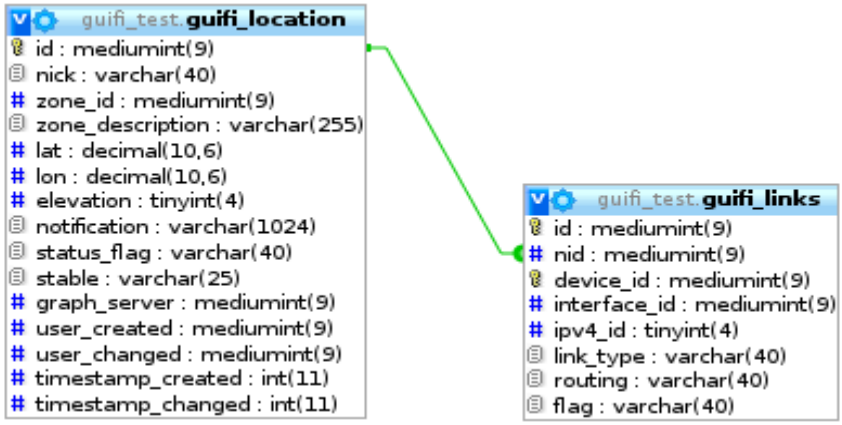


Ilustración 25: Diseño existente de la base de datos (Nodos, Enlaces).

Las siguientes entidades son de sólo lectura y de escaso volumen, por lo tanto se ha decidido incluirlas directamente en el código fuente.

- Proveedores
- Capas
- URL

La entidad “nodos visibles” es altamente volátil (el proceso Actualizar Nodos vacía el almacén en cada movimiento del mapa), de forma que no necesita persistencia de sesión. Su volumen se podría calificar de bajo (unos 300 nodos simultáneos).

Por lo tanto esta entidad se ha decidido convertirla en un almacén temporal en el cliente.

3.4. MODELO DE DISEÑO

En este apartado concretaremos más el modelo de diseño del software con información adicional. Ilustraremos partes del sistema mediante información textual y diagramas.

Como hemos visto en el diseño de interfaces, tenemos 3 componentes en nuestro sistema que son más o menos independientes: el **panel lateral**, el s**elector de capas** y la **capa de nodos dinámicos**.

El **panel lateral** no es más que un objeto contendor de elementos HTML. La primera función que realiza cuando se construye el objeto es verificar que se le han pasado parámetros que tienen sentido. Dichos parámetros no son HTML en bruto, sino una descripción de la estructura básica del menú. Después con estos datos construye el panel. Se puede acceder a los elementos INPUT del panel mediante una variable miembro, para poder conectar eventos entre ellos y otras partes del sistema.

El panel controla internamente el botón de mostrar / ocultar panel.

El formato de los parámetros es un objeto que contendrá las siguientes propiedades:

- "***forms***" (forzosamente 1 elemento)
  - **nombre de la sección** (0 a n): identificador único.
    - "***name***" (0 a 1): cadena con el título de la sección que se mostrará.
    - "***tooltip***" (0 a 1, requiere "name"): cadena con la descripción de la sección que se mostrará al mantener el puntero del ratón encima del título.
    - "***type***" (0 a 1, requiere "list"): "radio", "check" o cualquier otra cadena que decidirá el tipo de los elementos INPUT en el formulario.
    - "***list***" (0 a 1): contiene una lista de objetos con las siguientes propiedades:
      - **nombre de la opción** (forzosamente 1): identificador único
      - "***name***" (0 a 1): cadena con la etiqueta que se verá a la derecha del botón de la opción.
      - "***tooltip***" (0 a 1): cadena con el mensaje que se verá al mantener el punter del ratón por encima de la opción.
      - "***default***" (0 a 1): true o false. Si es true la opción estará marcada al cargarse el panel.
      - "***disabled***" (0 a 1): true o false. Si es true la opción estará desactivada y no se podrá seleccionar.
      - "**extrahtml**" (0 a 1): cadena con código HTML que se pondrá detrás de cada elemento de la lista
- "***extrahtml***" (de 0 a 1): cadena con código HTML que se pondrá al final de todas las opciones del panel.

El objeto entonces ofrece una propiedad, que es a su vez un objeto, llamada "inputs" que contiene los elementos INPUT que se añadieron con éxito al panel, cuyos nombres son los identificadores únicos que usamos al crear el objeto.

El usuario de este objeto (normalmente el mismo que lo crea) es responsable de conectar los eventos que generan los elementos INPUT a otras partes del sistema.

El **selector de capas** es un objeto medianamente complejo diseñado en base al patrón de diseño MVC (Modelo-Vista-Controlador). En la variante de MVC usada aquí, el Controlador contiene al Modelo y a la Vista, y es el encargado de inicializar ambas.

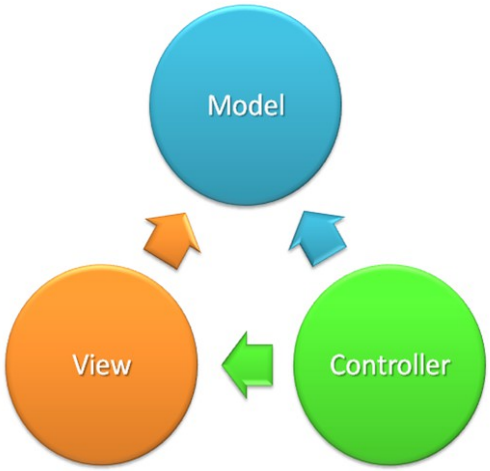


Ilustración 26: Esquema básico del patrón de diseño Modelo-Vista-Controlador ("Model-View-Controller")

La inicialización del Modelo por parte del Controlador consiste solamente en pasarle un subconjunto de los parámetros con los que se creó el Controlador. El Modelo representa la estructura de un menú (contenedor global, botones de menú principal, secciones, lista de botones, etc).

Debido a que los elementos del modelo se pueden anidar unos dentro de otros, se ha hecho que el Modelo mismo sea un elemento contenedor. Es decir, el objeto Modelo, puede ser a su vez un elemento de Modelo. De esta forma, si quisiéramos, podríamos anidar Modelos dentro de otros, con lo que podríamos combinar menús con otros, etc.

La Vista observa al Modelo y se actualiza siempre que éste cambia mediante el envío de eventos. El Controlador selecciona qué Vista usar (vista por defecto, vista para pantallas pequeñas, etc).

Desde fuera del objeto, se puede pedir a la Vista que se muestre u oculte, o conectar eventos para actualizar el Modelo, o añadir elementos al Modelo, o añadir observadores externos al Modelo, etc. Esto hace que este control sea muy configurable.

A continuación se explica el formato de los parámetros que se le pueden pasar al objeto selector de capas (LayerSwitcher):

- "***menu***" (forzosamente 1 elemento)
  - "***name***" (0 a 1): cadena con el nombre del elemento
  - "***title***" (0 a 1): cadena con la descripción del elemento
  - "***type***" (forzosamente 1 elemento): cadena con los valores posibles: "menulist", "radiolist", "checklist", "menu", "radio", "check", "extrahtml". Los objetos de tipo "*FOO*list" sólo pueden tener hijos de tipo "*FOO*" y "extrahtml".Los elementos de tipo "FOO" sólo pueden ser insertados en padres de tipo "*FOO*list", excepto "extrahtml" que puede ser insertado donde sea.

- “***list***” (0 a 1): contiene una lista de objetos con las mismas propiedades que “menu” (con lo que se pueden anidar objetos sin límite).
- “***selected***” (0 a 1): true o false. Si está a true el elemento se seleccionará al construirse el control.
- “***enabled***” (0 a 1): true o false. Representa si se puede seleccionar el elemento. Si se pone a false no se podrán realizar operaciones sobre el elemento hasta que se ponga a true.
- “***extrahtml***” (0 a 1): cadena con código HTML. Require type=”extrahtml”. La Vista puede decidir mostrar este código directamente como representación del elemento. El Modelo almacena este tipo de elemento, pero no produce eventos.

Cuando se selecciona un elemento de tipo “menu” o “radio” se deseleccionan sus hermanos. Cuando se selecciona un elemento de tipo “check” se seleccionan todos sus hijos.

La Vista utiliza los diferentes tipos de elemento (“check”, “radio”, “menu”, etc) para dibujar diferentes elementos HTML.

La Vista tiene una API con 2 funciones, “build” para construir los elementos HTML, y “toggle” para mostrar u ocultar la vista.

El Modelo tiene una API con las siguientes funciones:

- **attachChild**: para “colgar” un hijo de este elemento.
- **removeChildByName**: para eliminar a un hijo dado su nombre.
- **select**: para seleccionar el elemento.
- **unselect**: para deseleccionar el elemento.
- **selectAllChildren**: para seleccionar a todos los hijos
- **notifyAllSelections**: dispara el evento onSelect con el estado actual de selección para sí mismo y todos sus hijos. Se puede especificar que sólo se dispare el evento para aquellos elemenos que están seleccionados.
- **notifyChildrenSelections**: dispara el evento onSelect con el estado actual de selección para todos sus hijos. Se puede especificar que sólo se dispare el evento para aquellos elemenos que están seleccionados.
- **toString**: para transformar a cadena el elemento. La cadena resultante es “[MenuData nombre_del_elemento]”.

Eventos:

- **onSelected**: se dispara cuando el estado de selección del elemento cambia.
- **onChildSelected**: se dispara cuando se ha seleccionado un hijo del elemento.
- **onNewChild**: se dispara cuando se ha añadido un nuevo hijo al elemento.
- **onParented**: se dispara cuando se ha asignado un padre al elemento.
- **onChildRemoved**: se dispare cuando se ha eliminado a un hijo del elemento.

La **capa de nodos dinámicos** consiste simplemente en una función que hace una petición AJAX al servidor solicitando la lista de nodos dentro de una región.

Cuando llegan los datos, se recorre la lista de nodos y se añaden al mapa mediante funciones de la API de Google, configurando su color, forma, etc.

Cuando el mapa se mueve, o se cambia el zum, etc. se vuelve a repetir el proceso, borrando antes los nodos antiguos.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

ÍNDICE

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS..... 54
4.1. INTRODUCCIÓN..... 54
4.2. DECISIONES DE IMPLEMENTACIÓN..... 54
4.2.1. ESTRUCTURA DE ARCHIVOS..... 54
4.2.2. ORGANIZACIÓN DE TRABAJO..... 55
4.2.3. RENDIMIENTO..... 57
4.2.4. LIMITACIONES..... 58
4.3. PRUEBAS DE FUNCIONAMIENTO..... 59

4.1. INTRODUCCIÓN

La fase de implementación de un sistema consiste en transformar la especificación del diseño en un programa completo y funcional. Hasta ahora sólo habíamos trabajado con modelos, pero ahora se trata de usar los modelos como referencia para producir el programa final.

Aunque idealmente la implementación no se deba comenzar antes que el análisis y el diseño, lo cierto es que a veces es necesario implementar pequeños prototipos antes de estar seguros de que es posible programar ciertas funciones. En estos casos, es necesario volver atrás y modificar los modelos de análisis y diseño para reflejar los cambios.

4.2. DECISIONES DE IMPLEMENTACIÓN

4.2.1. ESTRUCTURA DE ARCHIVOS

Para añadir las nuevas funcionalidades usé la misma estructura de archivos que tenía el módulo Guifi original.

Los archivos nuevos o modificados fueron los siguientes:

- **guifi.install**: archivo existente. Modificado para añadir actualizaciones de rendimiento a la base de datos (solamente índices nuevos). Dichos cambios aparecen como una nueva actualización del módulo Guifi al administrador de la instalación de Drupal.
- **guifi_spatial.inc.php**: archivo nuevo. Su función es proporcionar una manera para que una aplicación externa solicite la lista de nodos dentro de una región. La aplicación sólo tiene que hacer una petición GET con la ruta de Drupal, seguido de “guifi/spatial/” más los parámetros necesarios separados por “/”, que son: el tipo de elemento(“nodes” o “links”), el estado (“all”, “air”, “cable”, “client” ó “wds”), el estado (“all”, “plan”, “test”, “nodrop”,

"Reserved", "Planned", "Building", "Testing", "Working" ó "Dropped") y las coordenadas de las esquinas de la región. Por ejemplo, lo normal es obtener los nodos de todos los estados considerando sólo enlaces inalámbricos para el recuento de enlaces, así que haríamos: "http://DOMINIO/guifi/spatialsearch/nodes/air/all/-90.0/-180.0/90.0/180.0". Esto nos devolvería una cadena JSON con información variada sobre todos esos nodos.

- **guifi.module**: archivo existente. Modificado solamente para incluir el archivo "guifi_spatial.inc.php" y para declarar que todas las llamadas a la ruta "guifi/spatial" se redirijan a una función declarada dentro de ese archivo.
- **js/wms-gs-2_0_0.js**: archivo existente. Modificado para incluir todos los elementos reutilizables del programa del cliente, como controles personalizados, nuevas capas, etc. Una persona que quisiese tener una aplicación de mapas similar a la de Guifi sólo tendría que incluir este archivo en su página web, añadir los controles al mapa y configurarlos, independientemente de la API usada (Google Maps, OpenLayers, etc). Todos los controles se han diseñado para que se puedan configurar en el momento de su creación con una simple cadena JSON.
- **js/guifi_gmap_zone.js**: archivo existente. Este archivo se incluye cuando cargamos una página de una zona en Guifi.net. Su única función es usar las funciones del archivo anterior para personalizar el mapa, según lo que un usuario espera ver cuando entramos en una zona.

4.2.2. ORGANIZACIÓN DE TRABAJO

Para organizar mi trabajo, aparte de herramientas como editores de texto, usé un programa de control de versiones llamado Git. Fue la elección lógica, ya que el módulo Guifi se encuentra ya en Gitorius.

Decidí empezar clonando no el repositorio original, sino un clon de éste en el cual se había portado la parte de mapas de la versión 2 de la API de Google (obsoleta y a punto de cesar el soporte) a la versión 3. Ese repositorio estaba previsto que se integrara en el repositorio original en breve, y de hecho así ha sido durante el transcurso del proyecto.

De mi repositorio en Gitorious hice 2 clones, uno en mi ordenador local, y otro en el servidor de prueba (alzina.act.uji.es).

La forma de trabajar fue la habitual con Git: crear una rama por cada "feature" e integrarlas en la rama principal periódicamente. Estas ramas locales no se suben a los repositorios públicos. Como rama principal para todo el proyecto usé la rama "googlev3".

En el ordenador local hice una instalación completa de Guifi (MySQL, Apache, Drupal) para poder hacer pruebas rápidas, y usé el servidor de pruebas para probar más a fondo el código que consideré más estable. Para actualizarlo, bastaba con hacer "git pull" a mi repositorio público.

Un organigrama de todo el montaje puede verse aquí:

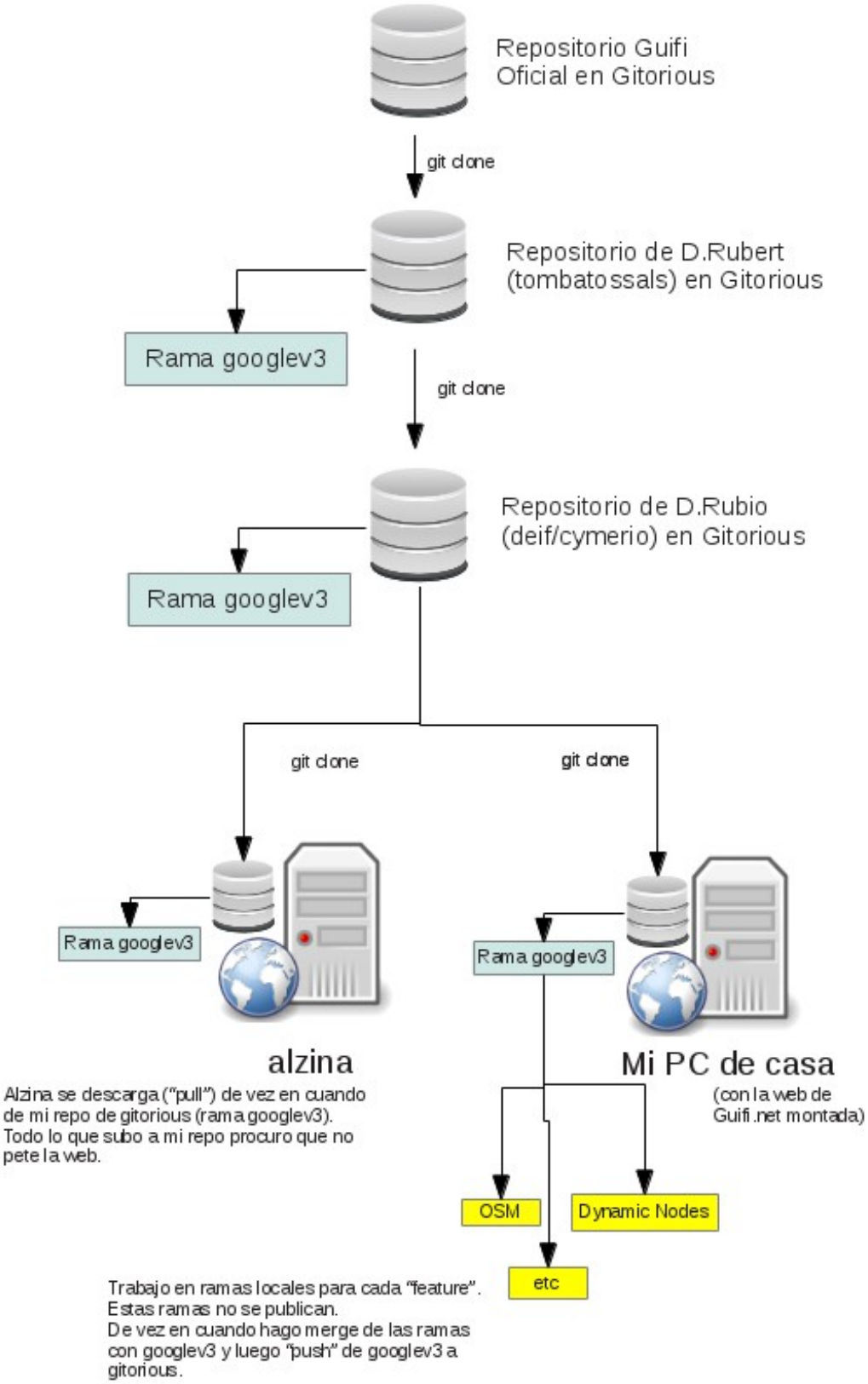


Ilustración 27: Organigrama de relaciones entre repositorios Git

En la siguiente figura puede verse parte del árbol de versiones generado al final del proyecto:

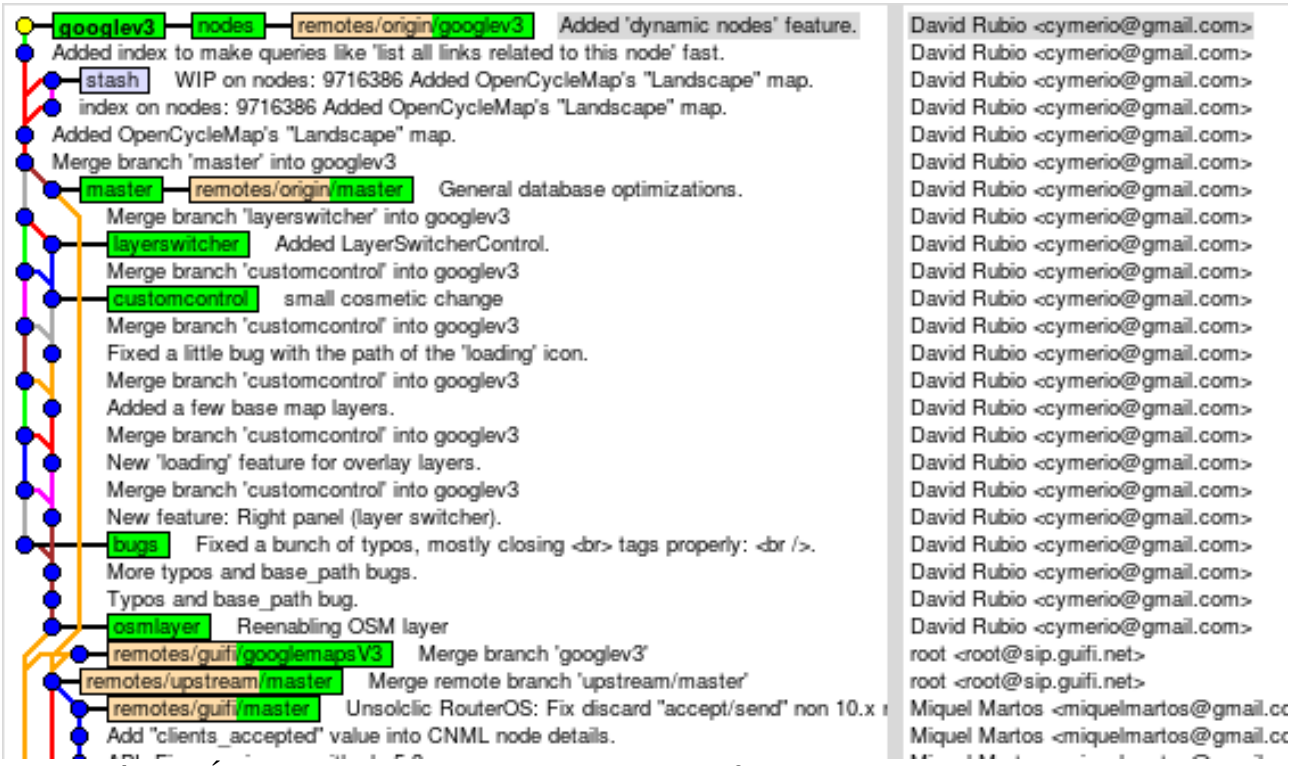


Ilustración 28: Árbol de versiones del repositorio local (fragmento)

4.2.3. RENDIMIENTO

Uno de los problemas detectados en el transcurso de la implementación fue que el rendimiento del módulo Guifi distaba de ser el óptimo.

El descubrimiento se originó investigando una manera óptima de recuperar los nodos dentro de una región, pero pronto pude ver que los problemas de rendimiento eran generalizados.

Haciendo uso de la orden EXPLAIN de MySQL pude perfilar algunas consultas y me di cuenta de que añadiendo unos pocos índices se podía mejorar el rendimiento de algunas de ellas considerablemente. Algunas de ellas pasaron de tardar segundos a milisegundos.

Aunque aumentar el rendimiento del sistema inicial no estaba dentro de los objetivos del proyecto, el cambio era tan mínimo que no pude evitar incluirlo. La creación de índices se incluyó dentro del archivo “guifi.install”, y para el administrador de Drupal los cambios se muestran como una actualización más del módulo. Los índices añadidos fueron los siguientes:

```
ALTER TABLE guifi_location ADD INDEX lat_index    USING BTREE (lat);
ALTER TABLE guifi_location ADD INDEX lon_index    USING BTREE (lon);
ALTER TABLE guifi_location ADD INDEX zcs_index    USING BTREE (zone_id,
timestamp_created, status_flag);
ALTER TABLE guifi_location ADD INDEX year_index   USING BTREE (timestamp_created);
ALTER TABLE guifi_location ADD INDEX status_index USING BTREE (status_flag);
ALTER TABLE guifi_devices  ADD INDEX nid_index    USING BTREE (nid);
ALTER TABLE guifi_links    ADD INDEX device_index USING BTREE (device_id);
ALTER TABLE guifi_services ADD INDEX zone_index   USING BTREE (zone_id);
ALTER TABLE guifi_zone     ADD INDEX master_index USING BTREE (master);
```

Estas mejoras todavía no han entrado en explotación en Guifi.net.

El problema de rendimiento más importante, dentro de los límites del proyecto, era que la recuperación de los nodos dentro de una región fuera eficiente.

Para ver dónde estaba el problema tenemos que pensar cómo se hacen internamente las consultas en MySQL:

Tenemos una serie de filas con dos columnas para representar la latitud y la longitud, y queremos buscar las filas que estén dentro de un rango, para la latitud y para la longitud. La consulta podría ser algo como "`SELECT * FROM guifi_location WHERE lat BETWEEN -90 and 90 AND lon BETWEEN -180 and 180`".

¿Qué es lo que hace MySQL cuando no hay ningún índice? Primero recorre todas las filas comprobando que la latitud esté dentro de los límites, y después vuelve a recorrer las filas seleccionadas haciendo lo mismo para la longitud, y las devuelve. En un caso como el del ejemplo, en el que el rango es el máximo posible, recorrería todos los nodos dos veces.

Si añadimos un índice a la latitud, por ejemplo, el primer recorrido se reduce a un par de búsquedas sobre un array ordenado (log n).

Podemos intentar añadiendo dos índices, pero MySQL sólo usará uno. MySQL sólo usa un índice por consulta, en cualquier consulta, ya que una vez seleccionadas unas filas por un criterio de una columna ordenada, esos resultados estarán desordenados para el segundo criterio, con lo que el segundo índice ya no nos vale.

Podemos intentar añadir un índice sobre dos columnas combinadas, pero estaríamos en las mismas. La ordenación se haría sobre la primera columna, y luego, en el caso de dos filas con el mismo valor en la primera columna, se ordenaría usando la segunda columna.

Por lo tanto se añadieron 2 índices, aunque en la práctica solo se llegue a usar uno, uno a la latitud y otro a la longitud. Será MySQL quien decida en cada momento qué índice usar. Se puede forzar a que MySQL analice las tablas con la órden ANALYZE TABLE para averiguar cuál es el índice mejor.

4.2.4. LIMITACIONES

Al haber basado el desarrollo sobre la API de Google Maps, el sistema desarrollado comparte varias de sus limitaciones.

Un ejemplo está en los eventos del mapa que pueden ser detectados. Google Maps no tiene un evento para detectar que una capa se está cargando, por lo que no se puede mostrar un indicador de "cargando" de una manera fiable. Sí que dispone, sin embargo, de un evento para "capa cargada".

Para solventar esto, se hizo que el icono de "cargando" se mostrase sólo en las pocas ocasiones en las que podemos estar seguros de que se necesita cargar parte de una capa. Estas ocasiones son el cambio de zum (que invalida totalmente la capa) y el cambio de dimensiones del marco.

Nótese que mover la capa no es condición suficiente para requerir cargar nuevos datos. Se podría dar el caso extremo de una tesela más grande que el propio mapa, en cuyo caso al moverla unos pocos milímetros seguiríamos dentro de ella, y no necesitaríamos cargar otras.

La librería OpenLayers, por otra parte, sí que proporciona eventos de capa cargándose y capa cargada.

4.3. PRUEBAS DE FUNCIONAMIENTO

La parte del cliente se probó en varios navegadores (Firefox, Chrome).

La parte del servidor se probó manualmente haciendo pasar parámetros erróneos.

El sistema entero se instaló en un servidor de prueba (http://alzina.act.uji.es/maps), donde ha sido probado por varias personas.

CAPÍTULO 5: DESPLIEGUE

ÍNDICE

CAPÍTULO 5: DESPLIEGUE 60
5.1. INTRODUCCIÓN 60
5.2. SERVIDOR DE TESELAS DE OSM 60
5.2.1. INSTALACIÓN DEL SOFTWARE NECESARIO 62
5.2.2. CONFIGURACIÓN 64
5.2.3. PREDIBUJAR ALGUNAS TESELAS 69
5.2.4. PROBANDO LA INSTALACIÓN 71

5.1. INTRODUCCIÓN

El despliegue consiste en la instalación y configuración final del software necesario. En nuestro sistema, además del software desarrollado, se ha de instalar software externo, y se ha de configurar de una manera específica.

Supondremos que ya disponemos de una instalación de Guifi.net. Para más información sobre cómo obtenerla, se puede consultar el siguiente documento:

http://es.wiki.guifi.net/wiki/Preparando_el_entorno_de_desarrollo

La única desviación de ese manual es que el código del módulo Guifi se ha de obtener del siguiente repositorio: https://gitorious.org/~deif/guifi/deifs-tombatossalss-drupal-guifi, y la rama a usar es “googlev3”.

5.2. SERVIDOR DE TESELAS DE OSM

Uno de los objetivos del proyecto es poder cambiar entre proveedores de mapas. OpenStreetMap como tal, no es un proveedor de servicios: es un proyecto para elaborar una base de datos geográfica libre. El servicio que ofrecen en su página web principal es limitado en recursos, y por tanto solicitan a los usuarios que si van a usarlo extensivamente hagan una instalación local para no saturar un servidor que se costea mediante donaciones voluntarias.

El servicio que queremos replicar (servidor de teselas) tiene la siguiente estructura:

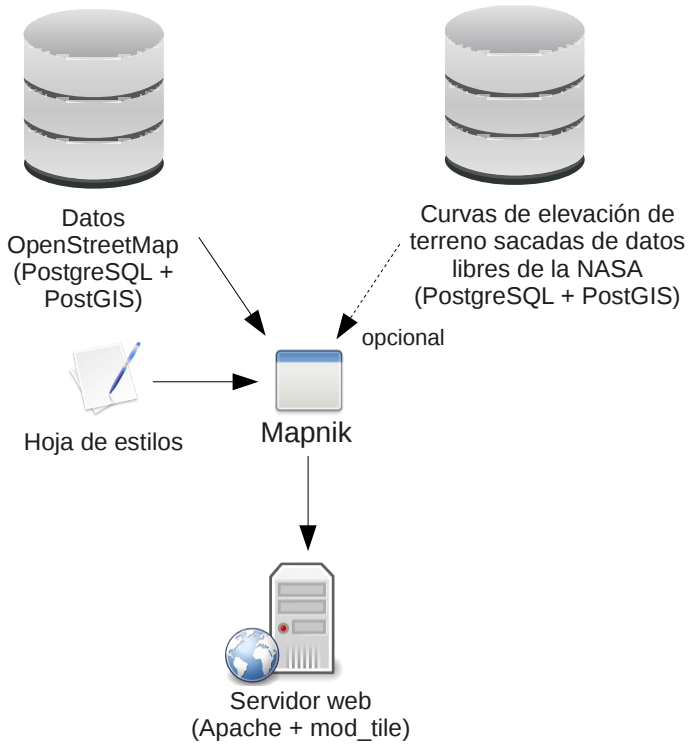


Ilustración 29: Estructura típica de un servidor de teselas

Es decir, tenemos una base de datos (PostgreSQL) donde guardaremos nuestros datos geométricos como puntos, líneas, polígonos, etc.; y un programa (Mapnik) capaz de coger esos datos y transformarlos en imágenes, de acuerdo a las reglas especificadas en una hoja de estilos. El servidor web se encarga de solicitar y enviar esas imágenes al usuario.

La única pega es que generar un mapa del planeta entero, a todos los niveles de zum, ocuparía unos 54 Terabytes, y tardaría varios meses en generarse, lo cual es inviable excepto para empresas como Google.

Incluso si dispusiéramos de los recursos necesarios seguiría siendo poco práctico, porque en OpenStreetMap aparecen nuevos datos cada día. Es decir, cuando terminásemos de generar todas las teselas, nuestro mapa estaría obsoleto.

Es por eso que normalmente se opta por otra estrategia, que es generar las teselas solo cuando se solicitan (estrategia “on demand”), y cachear las que se han solicitado al menos una vez. Periódicamente actualizaríamos la base de datos con nuevos datos y regeneraríamos solo las zonas que han cambiado. Con una estrategia así podemos tener un servidor de OpenStreetMap de todo el mundo con apenas 1 TiB (incluyendo caché), y solo de España con unos 5 GiB + caché. Otras opciones que aumentan el tamaño son incluir curvas de elevación de terreno (500 GiB extra) o generar y servir varios mapas con diferentes estilos (para esto debemos reservar más espacio de caché, aunque la base de datos no aumenta).

A continuación presentamos el procedimiento que se ha de seguir para disponer de un servidor de mapas (servidor de teselas) propio. Cuando exista alguna alternativa de configuración se mencionará en el texto.

5.2.1. INSTALACIÓN DEL SOFTWARE NECESARIO

Supondremos que partimos de una instalación típica de Debian (Debian 6: Squeeze), pero el procedimiento es extrapolable a cualquier sistema Linux.

Necesitaremos instalar la base de datos PostgreSQL junto con la extensión PostGIS:

```
aptitude install postgresql-8.4-postgis
```

Después necesitaremos instalar el programa "osm2pgsql", que va a ser el programa que usaremos para hacer la importación de datos de OSM a la base de datos. Si estamos en Debian es recomendable compilar la última versión, puesto que los paquetes están bastante desfasados.

Si vamos a usar el formato PBF (recomendado) debemos instalar antes la librería "libprotobuf-c0-dev". Como dicha librería no está en la distribución, tendremos que compilarla:

```
sudo apt-get install protobuf-compiler libprotobuf-dev libprotoc-dev
svn checkout http://protobuf-c.googlecode.com/svn/trunk/ protobuf-c-read-only
cd protobuf-c-read-only
./autogen.sh
make
sudo make install
```

Y a continuación obtenemos y compilamos "osm2pgsql":

```
svn co http://svn.openstreetmap.org/applications/utils/export/osm2pgsql/
cd osm2pgsql/
./autogen.sh
./configure
make
```

Si necesitamos más ayuda, hay más información en el siguiente enlace:

https://wiki.openstreetmap.org/wiki/Osm2pgsql#Installation

Ahora nos descargaremos la última versión de los datos de OpenStreetMap, o en su defecto un extracto de la zona que nos interese. Podemos usar el formato XML comprimido o el formato PBF. Si usamos el último la importación será mucho más rápida.

Hay muchos lugares de donde podemos descargarnos estos datos. Para ver una lista completa visitar este enlace:

https://wiki.openstreetmap.org/wiki/Planet.osm

Para descargar sólo España:

```
wget http://download.geofabrik.de/osm/europe/spain.osm.pbf # (300 MB)
```

Para descargar todo el mundo:

```
wget http://planet.openstreetmap.org/pbf/planet-latest.osm.pbf # (17 GB)
```

Una vez descargados los datos necesitaremos un programa que dibuje el mapa a partir de ellos, como por ejemplo Mapnik 2. Debian 6 no incluye este programa en su repositorio, así que tendremos que compilarlo. Es recomendable tener la versión más reciente de Mapnik 2, incluso obteniéndola del repositorio de desarrollo.

Mapnik 2.0.1 se puede descargar de aquí: https://github.com/mapnik/mapnik/downloads. Si queremos la última versión en desarrollo (recomendado) haremos:

```
git clone https://github.com/mapnik/mapnik.git
```

Instalaremos las dependencias para Mapnik 2:

```
sudo apt-get install g++-4.4 cpp libicu-dev libicu44 python2.6-dev libboost-system1.42-
dev libboost-filesystem1.42-dev libboost-iostreams1.42-dev libboost-thread1.42-dev
libboost-python1.42-dev libboost-program-options1.42-dev libboost-regex1.42-dev libxml2
libxml2-dev libfreetype6 libfreetype6-dev libjpeg62 libjpeg62-dev libpng12-0 libpng12-
dev libtiff4 libtiff4-dev libltdl7 libltdl-dev libproj0 libproj-dev libcairo2
libcairo2-dev python-cairo python-cairo-dev libcairomm-1.0-1 libcairomm-1.0-dev ttf-
dejavu ttf-dejavu-core ttf-dejavu-extra ttf-unifont postgresql-8.4 postgresql-server-
dev-8.4 postgresql-contrib-8.4 libgdal1-dev python-gdal postgresql-8.4-postgis
libsqlite3-dev subversion build-essential python-nose
```

Si hemos optado por descargar la versión en desarrollo de Mapnik necesitaremos una versión más reciente de Boost. Cuando se escribió esta memoria hacia falta la versión 1.47 o superior de Boost. En ese caso podemos instalar un Boost reciente así:

```
sudo apt-get install libbz2-dev
wget http://voxel.dl.sourceforge.net/project/boost/boost/1.50.0/boost_1_50_0.tar.bz2
tar xjvf boost_1_50_0.tar.bz2
cd boost_1_50_0
./bootstrap.sh
./b2 --with-thread --with-filesystem --with-iostreams –with-python \
     --with-regex -sHAVE_ICU=1 -sICU_PATH=/usr/local --with-program_options \
     --with-system link=shared toolset=gcc stage
sudo ./b2 --with-thread --with-filesystem --with-iostreams --with-python \
  --with-regex -sHAVE_ICU=1 -sICU_PATH=/usr/local --with-program_options \
  --with-system toolset=gcc link=shared install
sudo ldconfig
```

Para más ayuda compilando Boost: https://github.com/mapnik/mapnik/wiki/Mapnik2

Luego descargaremos el programa y lo compilaremos de la siguiente manera:

```
svn co http://svn.mapnik.org/trunk mapnik-trunk
cd mapnik-trunk
python scons/scons.py configure INPUT_PLUGINS=all OPTIMIZATION=3 \
SYSTEM_FONTS=/usr/share/fonts/truetype/
python scons/scons.py
sudo python scons/scons.py install
sudo ldconfig
```

Si se quiere cambiar el directorio de instalación se debe pasar la opción `PREFIX=”directorio”` y `PYTHON_PREFIX=”directorio”` (sin las comillas) a línea de “`python scons/scons.py configure`”.

Para más detalles se puede seguir esta guía:

https://github.com/mapnik/mapnik/wiki/DebianInstallation

También necesitaremos un servidor web para servir las imágenes. La opción ideal parecería ser “Nginx” o “Lighttpd” por su eficiencia al transferir imágenes estáticas, pero necesitamos algo más que transferir imágenes, y es un sistema para solicitar que se generen las imágenes cuando éstas no existen.

Uno de estos sistemas es Apache junto con la extensión “mod_tile”. Además dicha extensión proporciona beneficios adicionales como un cacheado eficiente, un mecanismo para reducir la carga al servidor aceptando clientes y poniéndolos en una cola, soporte para varias hojas de estilos, soporte para redibujar las teselas desfasadas, gráficas para monitorizar el estado del servidor, etc.

Por lo tanto instalaremos Apache y compilaremos “mod_tile”:

```
sudo apt-get install apache2
svn co http://svn.openstreetmap.org/applications/utils/mod_tile/
cd mod_tile
./autogen.sh
./configure
make
#
# Si llegados aquí hay errores de compilación
# hay que modificar 'gen_tile.cpp' y añadir los
# siguientes includes:
#     #include <mapnik/image_data.hpp>
#     #include <mapnik/image_view.hpp>
#     #include <mapnik/graphics.hpp>
#
sudo make install          # instala cosas en /usr/local
sudo make install-mod_tile # instala 'mod_tile.so' en el directorio de Apache
sudo ldconfig
```

5.2.2. CONFIGURACIÓN

Después de instalar todo el software, procederemos a configurarlo. No es necesario instalarlo todo primero y configurarlo después, pero aquí lo mostraremos así.

Primero optimizaremos la base de datos para el uso que le vamos a dar, es decir, grandes importaciones de datos y muchas consultas.

Editaremos el archivo '/etc/postgresql/8.4/main/postgresql.conf' y localizaremos y cambiaremos los siguientes valores dejándolos como sigue:

```
shared_buffers = 128MB
work_mem = 256MB
maintenance_work_mem = 256MB
checkpoint_segments = 20
autovacuum = off
```

Si PostgreSQL no arranca, probablemente haya que aumentar el tamaño máximo de la memoria compartida por el kernel:

```
sysctl -w kernel.shmmax=268435456
```

Para hacer este valor permanente hay que editar '/etc/sysctl.conf', añadiendo la línea 'kernel.shmmax=268435456' al final del archivo.

Se puede encontrar una guía más completa aquí:
https://wiki.openstreetmap.org/wiki/Mapnik/PostGIS#Tuning_the_database

El siguiente paso es crear la base de datos dentro de PostgreSQL. Para ello tenemos que ejecutar una serie de órdenes autentificados como el usuario 'postgres':

```
sudo -u postgres -i
createuser USUARIO
createdb -E UTF8 -O USUARIO gis
createlang plpgsql gis  # Solo para PostgreSQL < 9.1
```

Donde USUARIO es el nombre del usuario que hará el dibujado. Se puede usar un nombre de usuario existente o crear un usuario nuevo sólo para hacer estas tareas. En el servidor de prueba se creó el usuario 'gis' (no confundir con el nombre de la base de datos).

A continuación activaremos la extensión PostGIS para nuestra base de datos (siguiendo autentificados como 'postgres' o como el usuario USUARIO).

```
psql -d gis -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql
echo "ALTER TABLE geometry_columns OWNER TO USUARIO; ALTER TABLE spatial_ref_sys OWNER
TO USUARIO;"  | psql -d gis
```

Para más ayuda:
https://wiki.openstreetmap.org/wiki/Mapnik/PostGIS#For_Debian_testing_or_unstable_and_Ubuntu_10.4

Después importaremos los datos de OpenStreetMap que nos descargamos antes a nuestra nueva base de datos. Este paso puede tardar bastante dependiendo si nos descargamos el mundo entero (días o incluso una semana) o sólo una pequeña región como España (unos minutos).

```
psql -d gis -f /usr/share/postgresql/8.4/contrib/hstore.sql
osm2pgsql -m -d gis -s -U USUARIO –password --hstore planet-latest.osm.pbf
```

Hay que tener en cuenta que dicha importación es un procedimiento con pérdida de datos. Los datos se transforman durante la importación para que después el proceso de dibujado sea más rápido. La primera línea y el parámetro "--hstore" es opcional, y se usa para conservar algunos de estos datos que normalmente se perderían. Esos datos son datos no geométricos (que por tanto no se dibujan), como etiquetas poco usadas, pero pueden utilizarse con una hoja de estilos personalizada para dibujar ciertos objetos de diferente manera, o puede utilizarse para hacer consultas de tipo estadístico sobre el mapa (ej: ¿cuántos objetos hay con la etiqueta X?), etc.

Si queremos importar sólo una región de los datos que nos hemos descargado, podemos usar la opción "--bbox" para delimitar las coordenadas a importar.

Una vez con los datos en la base de datos, es hora de instalar las hojas de estilo que utilizará Mapnik para dibujar el mapa.

Estas hojas de estilo vienen en formato XML, y contienen las reglas que ha de seguir Mapnik para dibujar cada elemento. Por ejemplo, podemos establecer que las carreteras sólo se dibujen a partir de cierto nivel de zum, o que ciertos elementos se dibujen utilizando iconos en PNG, etc.

Dado que las hojas de estilo pueden hacer referencia a elementos externos, como imágenes o "shapefiles", colocaremos cada hoja de estilo en un directorio aparte con todos los archivos a los que haga referencia. La ubicación del directorio no es importante. El único requisito es que el usuario "USUARIO" tenga permisos de lectura.

Como ejemplo, vamos a instalar la hoja de estilos que se usa en la página principal de OpenStreetMap en el directorio "`/home/gis/stylesheets/mapnik`":

```
mkdir -p /home/gis/stylesheets
cd /home/gis/stylesheets
svn export http://svn.openstreetmap.org/applications/rendering/mapnik
cd mapnik
./get-coastlines.sh
./generate_xml.py --host localhost --user USUARIO --dbname gis \
  --symbols ./symbols/ --world_boundaries ./world_boundaries/ \
  --port 5432 --password CONTRASEÑA
```

Donde USUARIO es nuestro usuario de la base de datos, y CONTRASEÑA es la contraseña.

Para conprobar si todo está correcto podemos ejecutar el programa "`./generate_image.py`" el cual nos deberá generar el archivo "`image.png`" con un mapa del Reino Unido. Podemos cambiar la zona del mapa que se genera editando "`generate_image.py`" cambiando la variable "bounds" por otro valor, por ejemplo "`bounds = (-0.07336, 39.98887, -0.06081,39.99834)`" (para la zona de Castellón).

Hasta ahora tenemos los datos de OpenStreetMap en la base de datos, y tenemos una librería con la que se pueden dibujar mapas a partir de esos datos (Mapnik).

Lo siguiente es configurar el servicio web para servir las teselas. Este servicio web lo forman Apache, la extensión "mod_tile" y un demonio llamado "renderd" encargado de hacer el dibujado en sí (usando Mapnik). Dicho programa se instala por defecto al instalar "mod_tile".

Primero configuramos "renderd" editando "`/usr/local/etc/renderd.conf`" con el siguiente contenido:

```
[renderd]
socketname=/var/run/renderd/renderd.sock
num_threads=4
tile_dir=/var/lib/mod_tile ; DOES NOT WORK YET
stats_file=/var/run/renderd/renderd.stats

[mapnik]
plugins_dir=/usr/local/lib/mapnik/input
font_dir=/usr/local/lib/mapnik/fonts
font_dir_recurse=1

; Aqui "default" es el nombre que le estamos dando a nuestro mapa
; Pondremos tantas secciones como esta como hojas de estilo tengamos (maximo 10)
;    y daremos un nombre distinto a cada una: [mapa1], [blanco_y_negro], etc.
[default]
URI=/tiles/                             ; TESELAS ESTARAN EN "http://DOMINIO/tiles/z/x/y.png"
XML=/home/gis/stylesheets/mapnik/osm.xml  ; RUTA A NUESTRA HOJA DE ESTILO
HOST=alzina.act.uji.es                  ; NOMBRE DE DOMINIO DE NUESTRO SERVIDOR
;HTCPHOST=proxy.openstreetmap.org
```

Tenemos que asegurarnos sobre todo de modificar las variables URI, XML y HOST de acuerdo a las características de nuestro servidor.

Renderd no arranca automáticamente al arrancar el sistema. Para no tener que estar arrancando a mano este demonio cada vez que se reinicie el servidor, crearemos el archivo "`/etc/init.d/renderd`" con el siguiente contenido:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          renderd
# Required-Start:    $remote_fs
# Required-Stop:     $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Mapnik rendering daemon
# Description:       Mapnik rendering daemon.
### END INIT INFO

# Do NOT "set -e"

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="Mapnik rendering daemon"
NAME=renderd
DAEMON=/usr/local/bin/$NAME
DAEMON_ARGS=" -c /usr/local/etc/renderd.conf"
PIDFILE=/var/run/$NAME/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure that this file is present.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
    mkdir -p /var/run/renderd
    #chown www-data:www-data /var/run/renderd
    chown gis:gis /var/run/renderd
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started
        #start-stop-daemon --start --quiet --pidfile $PIDFILE --chuid www-data --exec $DAEMON --test
> /dev/null \
        start-stop-daemon --start --quiet --pidfile $PIDFILE --chuid gis --exec $DAEMON --test >
/dev/null \
                || return 1
        #start-stop-daemon --start --quiet --pidfile $PIDFILE --chuid www-data --exec $DAEMON -- \
        start-stop-daemon --start --quiet --pidfile $PIDFILE --chuid gis --exec $DAEMON -- \
                $DAEMON_ARGS \
                || return 2
        # Add code here, if necessary, that waits for the process to be ready
        # to handle requests from services started subsequently which depend
        # on this one.  As a last resort, sleep for some time.
}
```

```
                do_start
                case "$?" in
                        0) log_end_msg 0 ;;
                        1) log_end_msg 1 ;; # Old process is still running
                        *) log_end_msg 1 ;; # Failed to start
                esac
                ;;
          *)
                # Failed to stop
                log_end_msg 1
                ;;
        esac
        ;;
  *)
        #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
esac

:
```

```
        rm -f $PIDFILE
        return "$RETVAL"
}

#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
        #
        # If the daemon can reload its configuration without
        # restarting (for example, when it is sent a SIGHUP),
        # then implement that here.
        #
        start-stop-daemon --stop --signal 1 --quiet --pidfile $PIDFILE --name $NAME
        return 0
}

case "$1" in
  start)
        log_daemon_msg "Starting $DESC" "$NAME"
        do_start
        case "$?" in
                0|1) log_end_msg 0 ;;
                2) log_end_msg 1 ;;
        esac
        ;;
  stop)
        log_daemon_msg "Stopping $DESC" "$NAME"
        do_stop
        case "$?" in
                0|1) log_end_msg 0 ;;
                2) log_end_msg 1 ;;
        esac
        ;;
  #reload|force-reload)
        #
        # If do_reload() is not implemented then leave this commented out
        # and leave 'force-reload' as an alias for 'restart'.
        #
        #log_daemon_msg "Reloading $DESC" "$NAME"
        #do_reload
        #log_end_msg $?
        #;;
  restart|force-reload)
        #
        # If the "reload" option is implemented then remove the
        # 'force-reload' alias
        #
        log_daemon_msg "Restarting $DESC" "$NAME"
        do_stop
        case "$?" in
          0|1)
```

Nos aseguraremos de que las rutas que aparecen en ese archivo sean las correctas, y de que los permisos se otorgan al usuario correcto en la función "`do_start()`". En este ejemplo de script de arranque hemos hecho que el directorio "`/var/run/renderd`" pertenezca al usuario "`gis`", grupo "`gis`".

Ahora tenemos que configurar Apache para que cargue la extensión "mod_tile" y decirle dónde tiene que leer la configuración de este módulo.

La manera más rápida de hacerlo es crear el archivo "`/etc/apache2/conf.d/mod_tile`" con una sola línea:

```
LoadModule tile_module /usr/lib/apache2/modules/mod_tile.so
```

Y después añadir lo siguiente al archivo "`/etc/apache2/sites-available/default`", justo después de la línea que dice "`ServerAdmin webmaster@localhost`":

```
LoadTileConfigFile /usr/local/etc/renderd.conf
ModTileRenderdSocketName /var/run/renderd/renderd.sock
# Timeout before giving up for a tile to be rendered
ModTileRequestTimeout 3
# Timeout before giving up for a tile to be rendered that is otherwise missing
ModTileMissingRequestTimeout 30
```

No nos olvidemos de iniciar todos los servicios: postgresql, apache y renderd.

La caché de las teselas se colocará en "`/var/lib/mod_tile/NOMBREDEMAPA`", donde NOMBREDEMAPA es el nombre que hemos configurado en "`renderd.conf`", en nuestro caso "default".

Para más información sobre la configuración de "mod_tile" se pueden consultar los siguientes enlaces:

- https://wiki.openstreetmap.org/wiki/Mod_tile
- https://wiki.openstreetmap.org/wiki/HowTo_mod_tile
- http://svn.openstreetmap.org/applications/utils/mod_tile/readme.txt

5.2.3. PREDIBUJAR ALGUNAS TESELAS

Si ya se conoce con antelación las zonas del mapa que más se van a solicitar, es conveniente generarlas de antemano para que los primeros usuarios no tengan que esperar mucho. Además esto aligera la carga del servidor.

Siguiendo con la recomendación de usar la extensión de Apache "mod_tile" para cachear las teselas, vamos a usar algunos programas de ayuda que vienen con ella.

- **render_list**. Usaremos este programa después de la importación de datos inicial. El programa recibe una lista de las teselas que queremos generar y las genera incondicionalmente. Las teselas vienen especificadas por sus coordenadas y su nivel de zum. Dichas coordenadas no son pares latitud/longitud, sino que son índices numéricos que siguen una fórmula específica. Para detalles sobre la fórmula se puede consultar el siguiente enlace:

https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#coordinates_to_tile_numbers_2.

- **render_old**. Podemos usar este programa para regenerar todas las teselas anteriores a la fecha de importación de los datos. Cuando hacemos una importación de datos, automáticamente se genera un archivo con la fecha de la importación. Este programa lee esa fecha y actualiza las teselas que son anteriores . Podemos elegir los niveles de zum que se verán afectados, y elegir una fecha distinta a la de la importación. El inconveniente de este programa es que actualiza todas las teselas anteriores a una fecha independientemente de si han cambiado o no.

- **render_expired**. Usaremos este programa para marcar una región como obsoleta, e iniciar el redibujado de toda esa zona en todos los niveles de zum. Este programa se puede usar conjuntamente con "osm2pgsql" para regenerar solo las teselas que han cambiado en la última importación. Para más detalles consultar el siguiente enlace: https://wiki.openstreetmap.org/wiki/Tile_expire_methods#... with tiles expiry list from osm2pgsql_--append_--expire-tiles_--expire-output_option

A continuación se presenta un script de ejemplo que genera varias zonas de España, en especial la zona de Levante y los alrededores de la ciudad de Castellón:

```
#!/bin/sh
##########################################
#
# Programa para prerenderizar algunas teselas
#
##########################################


# Aplicaciones para pre-renderizar:
#    render_list    = renderiza todas las teselas en coordenadas y zum especificados (siempre)
#    render_old     = renderiza solo las teselas existentes con fecha anterior a la última importación de datos
#    render_expired = renderiza las teselas especificadas por z/x/y, en todos los niveles de zum

# Convertir coordenadas lat/lon a coordenadas de tiles:
#    https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#coordinates_to_tile_numbers_2


NOMBREMAPA=default
SOCKET=/var/run/renderd/renderd.sock
HILOS=1
TILEDIR=/var/lib/mod_tile


# Primero renderizamos un poco del mundo entero
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 0 -Z 4 -s $SOCKET

# Luego un poco de España
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 5 -Z 5 -x 15 -X 16 -y 11 -Y 12 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 6 -Z 6 -x 30 -X 32 -y 23 -Y 25 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 7 -Z 7 -x 60 -X 65 -y 46 -Y 50 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 8 -Z 8 -x 120 -X 131 -y 93 -Y 101 -s $SOCKET

# Levante...
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 9 -Z 9 -x 253 -X 262 -y 188 -Y 198 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 10 -Z 10 -x 507 -X 525 -y 376 -Y 396 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 11 -Z 11 -x 1014 -X 1050 -y 753 -Y 793 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 12 -Z 12 -x 2029 -X 2101 -y 1506 -Y 1586 -s $SOCKET

# Castellón ciudad y alrededores (Grao, Vila-real, Almazora, etc.).
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 13 -Z 13 -x 4092 -X 4096 -y 3100 -Y 3103 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 14 -Z 14 -x 8185 -X 8193 -y 6200 -Y 6207 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 15 -Z 15 -x 16371 -X 16387 -y 12400 -Y 12414 -s $SOCKET
render_list -a -n $HILOS -t $TILEDIR -m $NOMBREMAPA -z 16 -Z 16 -x 32742 -X 32774 -y 24801 -Y 24829 -s $SOCKET
```

Este programa se puede usar como base para generar otras zonas. Nótese que solo es útil después de la importación inicial, y que más adelante se debería usar solamente el combo "osm2pgsql" + "render_expired".

5.2.4. PROBANDO LA INSTALACIÓN

Se puede verificar que el servidor funciona accediendo a la siguiente URL:

http://nombre_de_dominio/tiles/0/0/0.png

Si todo ha salido bien, se debería ver una imagen similar a ésta:

Podemos crear rápidamente una pequeña página web para poder navegar por el mapa recién creado:

```
<html>
  <head>
    <title>OpenLayers Demo</title>
    <style type="text/css">
      html, body, #basicMap {
          width: 100%;
          height: 100%;
          margin: 0;
      }
    </style>
    <script src="http://www.openlayers.org/api/OpenLayers.js"></script>
    <script>
      function init() {
           var options = {
                projection: new OpenLayers.Projection("EPSG:900913"),
                displayProjection: new OpenLayers.Projection("EPSG:4326"),
                units: "m",
                maxResolution: 156543.0339,
                maxExtent: new OpenLayers.Bounds(-20037508.34, -20037508.34,
                                                 20037508.34, 20037508.34),
                numZoomLevels: 18,
                controls: [
                        new OpenLayers.Control.Navigation(),
                        new OpenLayers.Control.PanZoomBar(),
                        new OpenLayers.Control.Permalink(),
                        new OpenLayers.Control.ScaleLine(),
                        new OpenLayers.Control.MousePosition(),
                        new OpenLayers.Control.KeyboardDefaults()

                    ]
            };
        map = new OpenLayers.Map("basicMap",options);
        var newL = new OpenLayers.Layer.OSM("Default", "/tiles/${z}/${x}/${y}.png",
{numZoomLevels: 19});
        map.addLayer(newL);
        map.zoomIn();
      }
    </script>
  </head>
  <body onload="init();">
    <div id="basicMap"></div>
  </body>
</html>
```

Se puede ver una demostración en el siguiente enlace: http://alzina.act.uji.es/map.html.

En la dirección http://DOMINIO/mod_tile tendremos una pequeña página de estadísticas. En nuestro servidor de pruebas es http://alzina.act.uji.es/mod_tile.

La salida es similar a ésta:

```
NoResp200: 317
NoResp304: 49
NoResp404: 0
NoResp503: 0
NoResp5XX: 0
NoRespOther: 0
NoFreshCache: 366
NoOldCache: 0
NoFreshRender: 0
NoOldRender: 0
NoRespZoom00: 0
NoRespZoom01: 1
NoRespZoom02: 16
NoRespZoom03: 19
NoRespZoom04: 13
NoRespZoom05: 16
NoRespZoom06: 13
NoRespZoom07: 22
NoRespZoom08: 52
NoRespZoom09: 68
NoRespZoom10: 53
NoRespZoom11: 53
NoRespZoom12: 20
NoRespZoom13: 20
NoRespZoom14: 0
NoRespZoom15: 0
NoRespZoom16: 0
NoRespZoom17: 0
NoRespZoom18: 0
NoRes200Layerdefault: 366
NoRes404Layerdefault: 0
```

CAPÍTULO 6: ACCESO A LA APLICACIÓN

La aplicación se puede ver en funcionamiento desde las siguientes URL:

- Aplicación de mapas mostrando una zona: http://alzina.act.uji.es/maps/castello
- Zona aumentada: http://alzina.act.uji.es/maps/ca/node/18668/view/map

La última versión del código fuente siempre se podrá encontrar en este repositorio, en la rama "googlev3":

https://gitorious.org/~deif/guifi/deifs-tombatossalss-drupal-guifi

Para clonarlo, ejecutar:

```
git clone git://gitorious.org/~deif/guifi/deifs-tombatossalss-drupal-guifi.git
git checkout googlev3
```

CAPÍTULO 7: CONCLUSIONES FINALES Y TRABAJO FUTURO

La experiencia, en general, ha sido positiva. He tenido que aprender mucho sobre una gran cantidad de tecnología.

Hay que mencionar que éste no ha sido el típico proyecto de gestión, y lo cierto es que eso mismo ha sido un alivio general. Pienso que existe gestión más allá de los bancos y la contabilidad, y que no hay la suficiente diversidad de proyectos en la asignatura.

En el transcurso de este proyecto he aprendido muchísimo sobre todo el software relacionado con mapas, y he descubierto un pequeño “mundillo” sobre este tema, que parece no acabarse nunca.

El hecho de que este proyecto haya sido sobre todo visual, me ha ayudado a ver muy fácilmente mi progreso, y eso mismo ha sido lo que más me ha animado a continuar.

No estoy muy de acuerdo con las metodologías típicas (en cascada, etc), y es que yo soy más partidario de metodologías ágiles, donde la frontera entre las fases de un proyecto es un poco más difusa, y donde se asumen los cambios continuos como algo natural. Aún así, seguir una metodología es importante sobre todo para aclarar ideas y para dejarlo todo documentado para el futuro (para otros y para uno mismo).

Espero continuar con este proyecto hasta la integración definitiva en Guifi.net, y espero ayudar también en una futura migración a OpenLayers.

Como ya he dicho antes, la librería OpenLayers sin duda tiene mucha más potencia y capacidad de configuración que Google Maps. Cuando programé el prototipo para este proyecto (Ilustración 2) no tardé más que dos o tres días para una versión básica. El motivo fue que usé OpenLayers, que ya viene preparado con una plétora de controles y funcionalidad, que sólo hay que configurar y añadir al mapa.

Cambiar de librería no significa tener que renunciar a los servicios de Google. OpenLayers es capaz de cargar las capas de Google sin mucho esfuerzo de configuración. Si a esto le unimos que es software libre, y que tiene una gran comunidad detrás, la elección está clara.

Esto último es cada vez más importante, pues los servicios que hace tiempo eran gratuitos, como Google Maps, parece que se están volviendo difíciles de mantener por las empresas y cada vez se busca más rentabilizarlos. Google por ejemplo, anunció hace poco que Google Maps iba a ser de pago para aquellas páginas con un elevado número de peticiones a su servidor. Un tiempo después, Google aligeró un poco los términos de la nueva licencia debido a las críticas por toda la red.+

Además del uso de librerías libres, también sería ideal usar datos libres, como los de OpenStreetMap, Catastro o las ortofotos del PNOA. Éstas últimas, en sus versiones más recientes, tienen una resolución similar a la que consigue Google Maps. De hecho las fotos que tiene Google se obtienen a menudo de empresas mapeadoras locales, ayuntamientos o de datos liberados por el Estado.

Uno de los inconvenientes de las fotos del PNOA, es que vienen en proyecciones a las que no estamos muy acostumbrados. Dichas proyecciones, aunque superiores objetivamente, nos extrañan

a cualquiera que no sea un geógrafo.

Para reproyectar estas fotos, y conseguir un mapa en la misma proyección que Google u OpenStreetMap, tenemos que usar un software que haga de mediador entre un servidor de mapas y el usuario. Este software es MapProxy.

Lo que hace es agregar varios servicios de mapas y presentarlos como uno, reproyectando las capas que hagan falta para que todas estén en la misma proyección. Además de eso, cachea los resultados para no cargar a los servidores de mapas.

Otra alternativa es usar GeoServer para cargar las fotos del PNOA, reproyectarlas a la proyección que queramos y servirlas como un servicio más. Aún así, necesitamos una forma de cachear los resultados, y aquí vuelve a intervenir MapProxy.

Ilustración 30: Experimentos con MapProxy y las ortofotos del PNOA

Últimamente he estado haciendo experimentos también con diferentes estilos para los mapas. Lo ideal para Guifi.net sería un mapa con tonos de colores muy claros, para poder presentar mucha

cantidad de información encima y que se siga pudiendo distinguir del fondo. Además, objetos como torres de comunicaciones (móviles), tendido eléctrico, etc. sería interesante destacarlos en el mapa. No necesitamos ver destacadas las carreteras, pero sí por ejemplo serían interesantes las curvas de nivel.

Para hacer todo ello, existe un programa llamado TileMill que facilita la creación de estilos nuevos para los mapas. El coste adicional de servir varios estilos de mapas es sólo de caché en disco, ya que los datos tienen la misma fuente: los datos de OpenStreetMap que tenemos en PostgreSQL.

Si queremos hacer pruebas en casa, tendremos que instalar antes PostgreSQL e importar una zona como hemos descrito antes para montar el servidor de OSM (una importación de España se hace rápido y es suficiente para hacer pruebas). Podemos partir de un estilo similar a OSM llamado OSM Bright.

Los resultados surgen rápidamente:

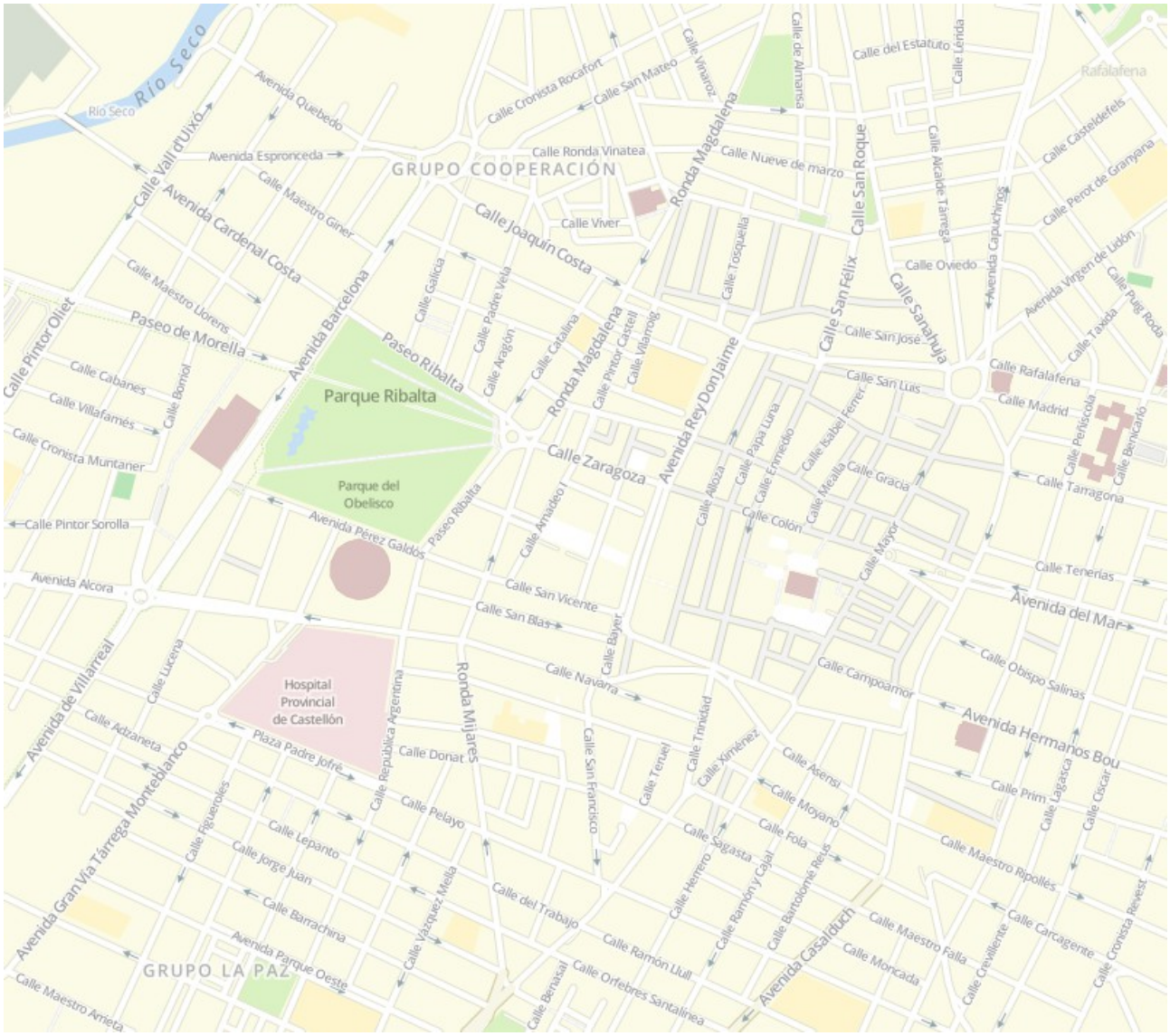


Ilustración 31: Mapa con estilo personalizado (desarrollado con Tilemill)

En el Capítulo 4: IMPLEMENTACIÓN Y PRUEBAS, apartado 4.2.3 Rendimiento hablábamos de un problema con el rendimiento para obtener una lista de nodos dentro de una región.

La solución óptima para el problema es hacer uso de las extensiones espaciales de MySQL. Dichas extensiones proporcionan un tipo de datos que representa un punto (un par de coordenadas) que se guarda internamente en un Árbol-R, que es un tipo de estructura en árbol optimizada para las búsquedas de puntos geométricos.

Para implementar esta solución consideré añadir una nueva columna de tipo espacial que representara el valor combinado de la latitud y la longitud, y crear un índice sobre ella. El código antiguo seguiría usando las columnas anteriores, y el código nuevo podría aprovecharse de esta ventaja. Para actualizar esta nueva columna pensé en añadir un disparador ("trigger") que la actualizara automáticamente cuando se hiciera una inserción.

El código podría quedar así:

```
ALTER TABLE guifi_location ADD COLUMN (loc Point NOT NULL COMMENT 'Stores the combined
value of lat and lon columns for faster searches using the spatial extensions');
UPDATE guifi_location SET loc = POINT(lat,lon);

DELIMITER |
CREATE TRIGGER latlon_to_loc BEFORE INSERT ON guifi_location
  FOR EACH ROW BEGIN
    SET NEW.loc = POINT(NEW.lat,NEW.lon);
  END
|
CREATE TRIGGER latlon_to_loc BEFORE UPDATE ON guifi_location
  FOR EACH ROW BEGIN
    SET NEW.loc = POINT(NEW.lat,NEW.lon);
  END
|
DELIMITER ;

ALTER TABLE guifi_location ADD SPATIAL INDEX loc_index (loc);
```

Desgraciadamente Drupal 6 no soporta la creación o acceso a columnas de tipo espacial directamente, sino que deja la responsabilidad al programador de escribir código portable a todas las bases de datos. Recordemos que Drupal está pensado para poder funcionar en un gran número de bases de datos.

Las extensiones espaciales no son como el SQL estándar, y son muy diferentes entre bases de datos, con lo que tendría que escribir una consulta diferente para cada base de datos, y algunas de ellas ni siquiera tienen un tipo de datos espacial.

Por estas razones descarté esta solución como inviable, pero Drupal 7 tiene soporte básico de tipos de datos espaciales, con lo que esta opción queda abierta al futuro.

En conclusión, una experiencia positiva, y un futuro que ofrece mucho potencial para las mejoras.

CAPÍTULO 8: APÉNDICE

ÍNDICE

CAPÍTULO 8: APÉNDICE..... 78
8.1. GLOSARIO..... 79
8.2. BIBLIOGRAFÍA..... 82

8.1. GLOSARIO

Borrado (nodo)
Estado de un nodo o enlace. Representa nodos o enlaces que se han declarado como eliminados de la red. Este estado no es muy habitual, ya que los nodos en desuso simplemente se suelen abandonar sin que quede ningún tipo de constancia en la web. Esto ha provocado que se tomen otras medidas para monitorizar los nodos más allá de lo que su dueño especifique en la web.

Carto
Lenguaje de hojas de estilo para mapas, parecido a CSS, desarrollado por MapBox para su uso con el programa TileMill.

En construcción (nodo)
Estado de un nodo o enlace. Representa nodos o enlaces que están siendo instalados o configurados y pueden no ser totalmente funcionales (o nada en absoluto).

En pruebas (nodo)
Estado de un nodo o enlace. Representa nodos o enlaces que ya se han terminado de instalar y ya funcionan, pero aún están en período de pruebas, por lo que la configuración aún puede cambiar.

Enlace
Conexión por cable o inalámbrica entre dos dispositivos. Lo habitual es conectar los dispositivos de un mismo nodo por cable, y los de nodos diferentes de manera inalámbrica (mediante ondas de radio). A efectos de visualización en un mapa, las conexiones dentro de un mismo nodo no se consideran, por estar en la misma ubicación. Así pues, en adelante usaremos una simplificación del concepto de enlace diciendo que es una conexión entre dos nodos.

Enlace troncal
Enlace entre dos supernodos.

Estado
Los nodos y los enlaces se pueden clasificar de varias formas según su estado de funcionamiento o intención. Los estados válidos son seis: Proyectado, Reservado, En construcción, En pruebas, Operativo y Borrado. Internamente se guardan con sus nombres en inglés: Planned, Reserved, Building, Testing, Working y Dropped. Incidentalmente, cada nombre en inglés empieza por una letra distinta.

Extensiones espaciales
También llamadas “spatial extensions”, son una extensión de MySQL para el soporte eficiente de tipos de datos geográficos y funciones que operan sobre ellos. No es tan completa como PostGIS, y de hecho algunas pruebas sugieren que su rendimiento puede ser un 50% de ésta. Como ventaja se puede destacar que la extensión viene instalada por defecto en MySQL. En sentido amplio, se usa para denominar a cualquier extensión de cualquier base de datos que soporta tipos de datos geográficos.

GIS
Siglas de “Geographic Information System” o Sistema de Información Geográfica. Es un término bastante amplio que abarca muchas de las aplicaciones usadas en este proyecto.

Google Maps

Servicio gratuito, pero no libre, de visualizado de mapas, de la empresa Google Inc. Este servicio tiene una API pública a la que se puede acceder mediante una librería que proporciona Google. La versión 2 de esta API está a punto de caducar y por lo tanto los programas que la usen dejarán de funcionar. La versión en uso actualmente es la 3. Google anunció hace poco que su servicio va a dejar de ser gratuito para páginas web con muchas visitas.

Guifi.net

Proyecto colaborativo para crear una red informática libre, abierta y neutral. Se pronuncia "Guifinet" o "Güifinet" indistintamente. Su nombre denomina también a la fundación Guifi.net y al módulo de Drupal llamado simplemente "Guifi".

gvSIG

Programa de usuario para la gestión geográfica. El programa permite cargar capas de múltiples servicios. Se puede usar para comprobar que un servidor WMS o de teselas funciona.

IGN

Siglas de Instituto Geográfico Nacional. Relacionados: PNOA.

Leaflet

Librería en Javascript para generar mapas en el lado del cliente. Tiene licencia libre y una pequeña comunidad. El principal desarrollador trabaja para la empresa CloudMade. Está diseñado para ser ligero y está indicado para dispositivos móviles.

LiDAR

Acrónimo del inglés *Light Detection and Ranging* o *Laser Imaging Detection and Ranging.* Es una tecnología parecida al radar, pero con láser, para calcular distancias. De especial interés para las aplicaciones geográficas es su uso para cálculos de elevaciones de terreno. Los datos LiDAR tienen extrema precisión. El IGN ha realizado varios vuelos LiDAR en la península en los últimos años y se espera que los datos estén disponibles pronto, con una resolución aproximada de 1 píxel por 2 metros cuadrados. Una resolución de ese tipo sería muy útil para el cálculo de líneas de visión entre antenas.

Mapbox

Empresa que basa su modelo de negocio en el alojamiento de mapas. Ofrecen dos servicios de teselas gratuitos con datos de OSM. También son los creadores de la herramienta Tilemill para el diseño de mapas vistosos.

Mapnik

Programa para dibujar ("renderizar") mapas a partir de datos en una base de datos y una hoja de estilos en XML. Su objetivo es generar mapas visualmente agradables. Está escrito en C++ pero también se puede usar desde Python.

Mapproxy

Aplicación que se coloca entre un servidor de mapas (WMS, servidor de teselas, etc) y el servidor web, para cachear las peticiones , agrupar servicios y realizar operaciones intermedias, como reproyecciones.

Nodo

Ubicación geográfica puntual donde se halla o se prevé instalar uno o más dispositivos de red (usualmente antenas). Un usuario del sistema puede ser dueño (a efectos de gestión) de varios nodos, necesitando especificar al menos uno para poder hacer uso de la red.

Nodo cliente

Nodo que sólo dispone de un enlace hacia otro nodo. Si representáramos a la red como un grafo no dirigido, diríamos que un nodo cliente es un vértice de grado uno. La definición de nodo

cliente se puede extender para denominar así también a los nodos que aún no tienen enlaces (no conectados a la red).

OpenLayers
Librería en Javascript para generar mapas interactivos en el lado del cliente. Tiene licencia libre y una amplia comunidad. Su principal característica es la potencia y la capacidad de configuración. Se puede recompilar para eliminar partes que no se usen y así reducir el tamaño del código.

OpenStreetMap
Proyecto colaborativo para crear una base de datos geográfica libre. Aunque los datos se distribuyen con una licencia libre, los recursos de la organización son limitados, y el servicio de teselas que se ofrece en su página principal, aunque gratuito, está protegido ante abusos. Se recomienda que si se va a hacer un uso masivo, se duplique el servicio en servidores propios.

Operativo (nodo)
Estado de un nodo o enlace. Representa nodos o enlaces que ya están funcionando y disponen de su configuración definitiva. Es el tipo de nodo más común.

Ortofotografía
Fotografía con la perspectiva rectificada para no mostrar deformaciones. Las fotos que solemos ver en Google Maps o servicios similares son todas ortofotos hechas a partir de muchas fotografías aéreas o por satélite.

OSM
Abreviatura de OpenStreetMap. Puede referirse a la organización, la página web, los mapas, o los datos de los que proceden.

OSM Bright
Estilo de mapa de demostración para TileMill escrito en lenguaje Carto. Está diseñado para parecerse al estilo oficial de OSM, pero con colores claros (pastel).

PNOA
Siglas de Plan Nacional de Ortofotografía Aérea. Es un proyecto del IGN para generar y poner a disposición de cualquiera fotografías aéreas de todo el territorio nacional. Las ortofotografía son simlpemente fotografías con la perspectiva y otras deformaciones corregidas.

PostGIS
Extensión de PostgreSQL para el soporte eficiente de tipo de datos geográficos y funciones que operan sobre ellos.

Proyectado (nodo)
Éste es el estado inicial de un nodo o enlace. Representa una ubicación posible pero todavía sin instalación alguna. En el caso de los enlaces representa la intención de conectarse a otro nodo en un futuro. Su utilidad es principalmente la de ayudar a la planificación futura de la red. El el segundo tipo de nodo más habitual.

Reservado(nodo)
Estado de un nodo o enlace. Representa nodos o enlaces que solo están en funcionamiento durante determinados períodos de tiempo, pero que no obstante se les quiere asignar recursos fijos (principalmente direcciones IP).

Servidor de teselas.
Servidor de mapas que sólo sirve imágenes (sin metadatos) en una proyección fija, en resoluciones pequeñas, típicamente 256x256 ó 512x512. Están optimizados para servir un gran número de imágenes.

Shapefile

Formato de archivo propietario para almacenar datos espaciales. Estándar de facto para el intercambio de información geográfica entre Sistemas de Información Geográfica.

Supernodo

Nodo que dispone de dos enlaces o más hacia otros nodos, normalmente utilizando más de una antena y colaborando en el encaminamiento dinámico.

Tile server.

Véase “servidor de teselas”.

Tilemill

Programa en Java con interfaz gráfica para diseñar hojas de estilo para programas de dibujo (“renderizado”) de mapas. Usa un formato propio de hojas de estilo llamado Carto, pero puede exportar a XML entendido por Mapnik.

WMS

“Web Map Service”. Servicio web de mapas. Es un estándar definido por el Open Geospatial Consortium para servir imágenes y metadatos de mapas. El usuario accede al servicio mediante peticiones GET o POST, donde especifica la operación a realizar, como: consultar las operaciones disponibles, las proyecciones disponibles, obtener un mapa, etc. Son más versátiles que los servidores de teselas, pero menos eficientes cuando se quieren recuperar un gran número de imágenes de poco tamaño. Este tipo de servidor genera una imagen nueva en cada petición si no se utiliza algún mecanismo de caché.

8.2. BIBLIOGRAFÍA

1. Wikipedia
2. http://es.wiki.guifi.net/wiki/Preparando_el_entorno_de_desarrollo
3. Documentación de Drupal
4. Wiki de OpenStreetMap
5. Apunts d'Enginyeria del Programari de Gestió I (IG16), Cristina Campos Sancho, Reyes Grangel Seguer, Vicente Verde Pelcato