

Enfoque práctico

**Desarrollo de Sistemas de Información Geográficos sobre
una plataforma soberana.**

Ing. Antonio Membrides Espinosa



Aclaración: este documento no está terminado, es tan solo un borrador

**Habana,
1 de Junio del 2013**

Enfoque práctico, para el desarrollo de Sistemas de Información Geográficos sobre una plataforma soberana.

Este documento tiene como objetivo principal aglomerar un gran cumulo de información, que permita sentar las bases cognitivas para todos aquellos que se aventuren en el desarrollo de sistemas de información geográfico. Por tanto se desglosa el contenido a través de un conjunto de epígrafes que describen los elementos que fueron considerados como esenciales para adentrarse en este mundo, tomando como base la plataforma GeneSIG.

Para la confección del mismo fue imprescindible el aporte científico de un grupo de especialistas, los cuales por su experiencia en esta área temática jugaron un rol protagónico en la realización de este trabajo.



Nombre: Hermes Lázaro Herrera Martínez

Correo electrónico: hlherrera@uci.cu

Especialidad: Ingeniero en Ciencias Informáticas.

Institución: XETID, MINFAR

Desempeño: Desarrollador de tecnologías de software para sistemas de información geográficos, almacenamiento y representación espacial.

Arquitecto del desarrollo de sistemas de reconocimiento de patrones basados en técnicas de minería de datos.



Nombre: Eduanys Puerta Ramos

Correo electrónico: epuerta@uci.cu

Especialidad: Ingeniero en Ciencias Informáticas.

Institución: XETID, MINFAR

Desempeño: Desarrollador de tecnologías de software para la creación de sistemas de información geográficos. Certificado como desarrollador de lógica de negocio e interfaz de usuario. Se especializó en los sistemas para la generación de caché de mapas.

También es meritorio reconocer a todas aquellas personas que de una forma u otra aportaron su granito de arena con cada una de sus ideas y experiencias. Principalmente los profesionales y estudiantes vinculados a la línea GIS del centro SATD, tales como *Romanuel Ramon Antunez, Armando Batista Piñeda, Alejandro Rangel Valdivia, Isidro Morejón Machado*, entre otros tantos.

Índice

Introducción	6
1. Conceptos	7
1.1 Mapa	7
1.2 Leyenda	8
1.3 Escala	8
1.4 Proyección	8
1.4.1 Esquema de una proyección cónica	11
1.4.2 Esquema de una proyección azimutal, cenital o polar	12
1.4.3 Esquema de una proyección cilíndrica	12
1.4.4 La proyección Universal transversa de Mercator	12
1.4.4 Sistemas de referencia utilizados en ambientes SIG	13
1.5 Capa	14
1.6 Modelo de datos Raster	14
1.7 Modelo de datos Vectoriales	14
1.8 Puntos	14
1.9 Líneas	15
1.10 Poligonales	15
1.11 Información socioeconómica	15
1.12 Modelos Digitales del Terreno	15
1.13 Modelo Digital de Elevaciones	16
1.14 Open Geospatial Consortium	17
2. Elementos técnicos necesarios	18
2.1 SIG	18
2.2 OOP	18
2.3 PHP	19
2.4 JavaScript	19
2.5 PEAR	19
2.6 Ajax	19
2.7 CGI	20
2.8 Servidor Web	20
2.9 Servidor de Mapas	20
2.10 Servidor de Bases de Datos	2
3. Procedimiento de instalación	2
3.1 Servidor Web	2
3.1.2 Virtualización	2
3.1.3 Privilegios de ejecución	5
3.2 Servidor de Bases de Datos	7
3.2.1 Configuración	8
3.2.2 Creación de bases de datos espaciales	11
3.3 Servidor de Mapas	11
3.4 Complementarios	12
3.4.1 PGrooting	12
4. Mapserver	12
4.1 Mapfile	14
4.2 Modo CGI	22
4.2.1 Confeccionando un mapa básico	26
4.2.2 Inclusión de ficheros externos	26
4.2.3 Filtrado de datos y expresiones	30
4.3 MapScript	31
4.3.1 Cargando un mapa	31

4.3.2 Adicionando elementos de interacción	32
4.3.3 Confeccionando un mapa dinámico	39
4.3 SLD	43
4.4 Servicios de mapas	47
4.4.1 WMS	47
4.4.2 WFS	55
4.4.3 WCS	59
4.5 Catálogo de funciones	61
5. GeneSIG	62
5.1. Plugins y Coreplugins	67
5.2. Configuración	68
5.2.1 Cliente	68
5.2.2 Servidor	69
5.2.3 Adicionando módulos	69
5.2.4 Accediendo a las variables	71
5.2.4 Variables de Entorno	71
5.3 Infraestructura	72
5.4 Interfaces	74
5.4.1 Comunicación	75
5.4.2 Control de sesiones	75
5.5. Mecanismos de Comunicación	77
5.5.1 Comunicación Client.php con Smarty	78
5.5.2 Comunicación Client.js con Client.php	79
5.5.3 Comunicación Client.php con Server.php	85
5.5.4 Comunicación entre plugins	86
5.6 Controladores Frontales	87
5.6.1 Generador de URL	89
5.6.2 Ejecutor de peticiones al servidor	90
5.7 Acceso a Datos	91
5.7.1 Ejecutar un script SQL	91
5.7.2 Escape SQL	93
5.7.3 Extracción de información de las Layer	94
5.8 Mapscript desde dentro	95
5.9 Recursos	96
5.9.1 Base	96
5.9.2 Layer	100
5.9.3 Location	101
5.9.4 Query	105
5.9.5 Auth	110
5.9.6 Priolus	113
5.9.7 Manipulación de Mensajes	115
5.10 Tutorial	117
5.10.1 Descargando el código fuente	117
5.10.2 Creando un Proyecto	120
5.10.3 Adicionando un Plugin	125
5.10.4 Desarrollando un Plugin	131
5.11 Servicios	144
5.11.1 Modo de acceso	145
5.11.3 Elementos del contrato	146
5.11.3 Llamadas a procedimientos genéricos	147
5.11.4 Mapa	150
5.11.5 Navegación	152

5.11.6 Autenticación.....	154
5.11.7 Identificación	155
5.11.8 Tematizacion	157
5.11.9 Perfiles de visibilidad y altura	160
5.12 Integración con plataformas externas.....	162
5.12.1 Openlayer.....	163
5.12.2 Openlayer y ExtJs.....	166
5.12.3 Zeolides	170
5.13 Proyección para la versión 2.0.....	171
6. Openlayer.....	171
6.1 Elementos basicos	171
6.1.1 Tipos básicos de datos	174
6.1.2 La clase OpenLayers.Util	175
6.1.3 La clase OpenLayers.Map	176
6.2 Controles.....	177
6.2.1 La clase OpenLayers.Control.....	179
6.2.2 Utilización de los controles	179
6.2.3 Botones personalizados	181
6.3 Capas	183
6.3.1 La clase OpenLayers.Layer	183
6.3.2 Tipos de capas.....	185
6.9 Un enfoque diferente basado en HTML5.....	221
6.9.1 Elementos HTML5	221
6.9.2 Representación de datos espaciales	223
6.9.3 Clases capas de OpenLayers	224
6.9.4 Análisis Raster en OpenLayers.....	229
6.10 Conclusiones.....	232
7. Sistema para la generación de caché de mapas	232
7.1. Instalación del módulo de Apache2.0 para Python.....	233
7.2. Instalación de Tilecache v1.9	234
7.3. Configuración del servidor de teselas	234
7.3.1 Fichero de configuración tilecache.cfg	235
7.3.2 Cálculo de resoluciones y escalas para Tilecache	237
7.4 Generación manual de la caché de mapas	238
7.5 Generando la caché para Mapserver.....	239
7.5.1 Funcionalidad mapserver_seed.py	241
7.5.2 Ejecutando mapserver_seed.py	241
7.6 Tileindex	241
7.6.1 Configuración del fichero de *.map	242
7.7 Conclusiones	242
8 Manipulación de datos espaciales	243
8.1 Objetos GIS	244
8.1.1 Forma CANONICA vs ESTANDAR	245
8.2 Usar el estándar OpenGIS.....	245
8.2.1 SPATIAL_REF_SYS	246
8.2.2 GEOMETRY_COLUMNS.....	247
8.2.3 Crear una tabla espacial	247
8.3 Importar datos datos espaciales	247
8.4 Exportar datos espaciales	249
8.5 Índices	250
8.5.1 GIST	251
8.5.2 Empleo	251

8.6 Funciones.....	252
8.7 Migración de bases de datos en función del soporte de postgis	260
9. Herramientas complementarias.....	260
9.1 GDAL.....	261
9.2 OGR.....	261
9.3 Proj4.....	261
9.3.1 Reproyección de coordenadas georeferenciadas	261
9.3.2 Transformación de sistema de referencia de coordenadas	262
9.3.3 Cálculos de Azimut y distancia de problema inverso	262
9.4 Shp2img	263
9.5 Legend.....	263
9.6 Scalebar	263
9.7 Sortshp	263
9.8 Sym2imp	263
9.9 Shptree	264
9.10 Tile4ms	264
Glosario de términos.....	265
Bibliografía.....	265

Tabla de Ilustraciones

Ilustración 1 Modelos de proyecciones.....	11
Ilustración 2Constantes de transformación de SR, según ecuaciones de Molodensky.	13
Ilustración 3 Parámetros geodésicos asignados a WGS84	14
Ilustración 4 Ejemplo del ejemplo del raster para describir el terreno	14
Ilustración 5 Sistema de Referencia Espacial.	15
Ilustración 6 Modelo Digital de Elevación.	16
Ilustración 7 Representación tipo malla en vista isométrica	17
Ilustración 8 Representación en dos dimensiones por medio de tonos	17
Ilustración 9 Elementos de comparación entre servidores de mapas	1
Ilustración 10 Listado de paquetes relacionados con ‘posgres’ desde el <i>Synaptic</i>.....	8
Ilustración 11 Anatomía de Mapserver	14
Ilustración 12 Relación entre los objetos del mapfile	15
Ilustración 13 Estructura del proyecto basado en la inclusión de fichero externos.....	32
Ilustración 14 Vista del ejemplo2, basado en la generación de mapas estáticos	34
Ilustración 15 Estructura física del ejemplo para la creación de un mapa dinámico.....	40
Ilustración 16 Taxonomía del ejemplo para la generación y carga de SLD.....	44
Ilustración 17 Plataforma GeneSIG.....	62
Ilustración 18 Aplicacion demostrativa de CartoWeb	63
Ilustración 19 Arquitectura definida por CartoWeb.	64
Ilustración 20 Modos de comunicación Directo y SOAP	64
Ilustración 21 Modos de comunicación Multiclient.....	65
Ilustración 22 Modos de comunicación Composition of architectures	65
Ilustración 23 Elementos que intervienen en la arquitectura de CartoWeb.....	66
Ilustración 24 Entorno genérico para el despliegue de GeneSIG	66
Ilustración 25 Estructura física para un plugin de GeneSIG	68
Ilustración 26 Taxonomía de la plataforma	72
Ilustración 27 Estructura del directorio htdocs de GeneSIG	73
Ilustración 28 Interfaces definidas para el modulo cliente de un plugin o coreplugin	74
Ilustración 29 Interfaces definidas para el modulo servidor de un plugin o coreplugin	75
Ilustración 30 Secuencia de invocación de eventos por interfaces	78

Ilustración 31 Vista general de la GUI para GeneSIG.....	97
Ilustración 32 Vista del menu contextual sobre el controlador de capas	101
Ilustración 33 Interfaz para la gestión de capas seleccionables y/o consultables.....	106
Ilustración 34 Vista que represa la acción de consulta o identificación sobre el mapa ...	110
Ilustración 35 Vista del modulo para la autenticación de la plataforma	110
Ilustración 36 Modulo Priolus, aunque aparece con la denominación estructuras	114
Ilustración 37 Interfaz de visualiza de mensajes sobre la plataforma GeneSIG	116
Ilustración 38 Comando de consola para crear el espacio de trabajo.	117
Ilustración 39 Primer paso en la descarga basada en un SVN	118
Ilustración 40 Segundo paso en la descarga basada en un SVN	119
Ilustración 41 Tercer paso en la descarga basada en un SVN	119
Ilustración 42 Cuarto paso en la descarga basada en un SVN	120
Ilustración 43 Paso 1 en la creación de un proyecto.....	121
Ilustración 44 Paso 3 en la creación de un proyecto.....	122
Ilustración 45 Paso 4 en la creación de un proyecto.....	123
Ilustración 46 Paso 5 en la creación de un proyecto.....	124
Ilustración 47 Paso 7 en la creación de un proyecto.....	125
Ilustración 48 Paso 1 en función de adicionar un Plugin	126
Ilustración 49 Paso 2 en función de adicionar un Plugin	126
Ilustración 50 Paso 3 en función de adicionar un Plugin	127
Ilustración 51 Jerarquía del esquema organizacional de las herramientas en la GUI.....	127
Ilustración 52 Paso 4 en función de adicionar un Plugin	128
Ilustración 53 Paso 5 en función de adicionar un Plugin	129
Ilustración 54 Paso 6 en función de adicionar un Plugin	129
Ilustración 55 Paso 7 en función de adicionar un Plugin	130
Ilustración 56 Paso 8 en función de adicionar un Plugin	130
Ilustración 57 Accediendo al plugin Search a través del toolbar	131
Ilustración 58 Taxonomía del proyecto ejem.....	131
Ilustración 59 Estructura de la parte cliente del modulo Search	132
Ilustración 60 Emplear la herramienta de trazado del plugin Search	134
Ilustración 61 Especificar los datos del formulario para el plugin Search.....	137
Ilustración 62 Traseando el código cliente del plugin Search	138
Ilustración 63 Observar la respuesta del plugin Search para el formato json.....	138
Ilustración 64 Observar la respuesta del plugin Search para el formato xml	139
Ilustración 65 Estructura de la parte servidora del plugin Search.....	139
Ilustración 66 Mecanismo de comunicacion en modo SOAP para un plugin	145
Ilustración 67 Resultado arrojado por el consumo de servicio de mapa.....	152
Ilustración 68 Vista resultante de plugin Identificacion	156
Ilustración 69 Tematizacion de tipo Gráficas Dinámicas.....	159
Ilustración 70 Tematizacion de tipo Corocromático y Coropleta.....	160
Ilustración 71 Esquema teórico de un perfil de visibilidad y altura	161
Ilustración 72 Taxonomía del ejemplo #1 de integración	163
Ilustración 73 Ejemplo #1 de integración	164
Ilustración 74 Taxonomía del ejemplo #2: Integración basada en Openlayer y ExtJs	166
Ilustración 75 Vista del ejemplo #2: Integración basada en Openlayer y ExtJs	169
Ilustración 76 Caché de mapa por escala	239
Ilustración 77 Funcionalidad de Mapserver	239
Ilustración 78 Estructura de carpetas por escala	240
Ilustración 79 Estructura de paquetes por escala.....	240
Ilustración 80 Prototipo de cambio de sistema de referencia	262
Ilustración 81 Ejecución de calculo de problema inverso	262

Introducción

La Universidad de las Ciencias Informáticas a lo largo de los años ha adquirido gran experiencia en el desarrollo de grandes proyectos contribuyendo de una forma u otra en la formación de profesionales con grandes conocimientos en áreas de la informática. Con el desarrollo de la Plataforma Soberana GeneSIG, producto que une a equipos de desarrollo como el Centro UCIFAR y GeoCuba, se persigue fortalecer la experiencia en la realización de sistemas de información geográfica y abrir un espacio sólido en el mercado de aplicaciones de esta rama, reutilizando los componentes y funcionalidades para personalizar los productos en cualquier negocio que lo requiera. (Yoenis Pantoja Zaldívar, 2010)

La Plataforma GeneSIG es un producto encaminado a realizar la representación y análisis de información geográfica. Permitiendo adaptar el sistema a las características del negocio que lo requiera mediante la reutilización de sus componentes. Puede ser considerado como un Sistema de Información Geográfica Web único y extensible, basado en estándares OpenGIS que incluye funcionalidades operativas de las aplicaciones de esta tecnología. (Yoenis Pantoja Zaldívar, 2010)

El producto GeneSIG en su versión 1.5, se encuentra desarrollado sobre las siguientes tecnologías y herramientas:

- **Lenguajes de desarrollo:** PHP 5 y JavaScript.
- **Entornos Integrados de Desarrollo:** Zend Studio y Aptana.
- **Sistema Gestor de BD:** PostgreSQL
- **Sistemas Operativos:** Ubuntu 9.04 y Windows XP SP3
- **Control de Versiones:** RapidSVN y Tortoise 1.4.5
- **Sistema de Gestión de Proyecto:** DotProject
- **Herramienta de Modelado:** Visual Paradigm 3.5
- **Frameworks:** CartoWeb 3.0, ExtJs 3.2
- **Servidor de Mapas:** MapServer 2.4.6
- **Servidor Web:** Apache 2.5.6

Uno de los elementos claves en el desarrollo de GeneSIG y de obligatorio estudio para su futura comprensión, es el framework CartoWeb. Este marco de trabajo constituye la columna vertebral de dicha plataforma y todos sus mecanismos de diseño están orientados en función de la arquitectura que el mismo propone. Por tanto en los acápite que se describen a continuación se podrán percibir que se utiliza el término de CartoWeb en la descripción de la gran mayoría de los recursos que provee GeneSIG.

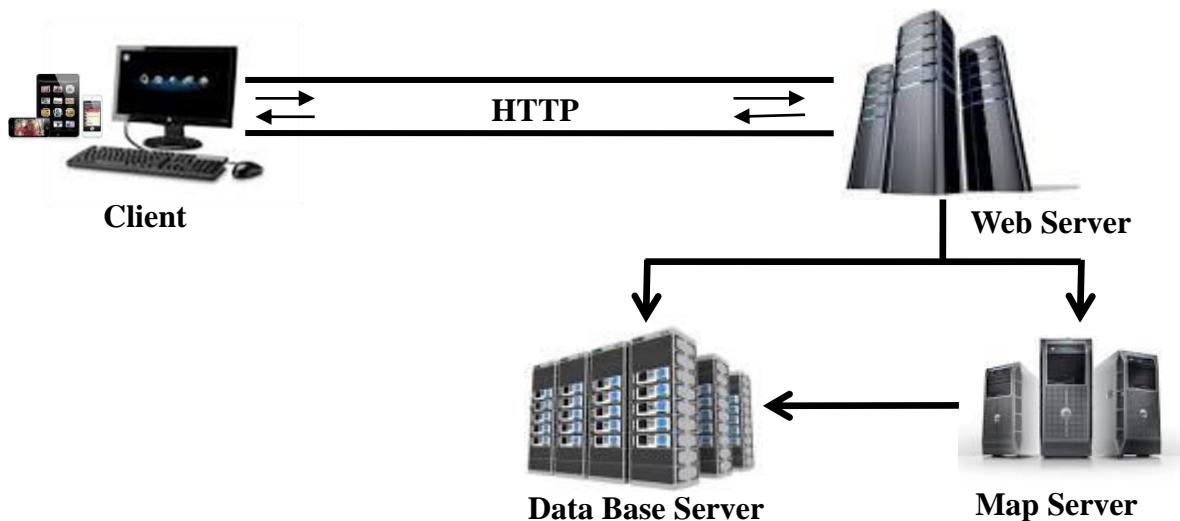
CartoWeb es un software de publicación WebGIS construida en PHP sobre UMN MapServer que explota la tecnología AJAX. Su característica más diferenciadora respecto a otros proyectos de clientes Web ligeros sobre MapServer, es que ofrece un *framework* que ha sido diseñado con una arquitectura modular y escalable, permitiendo separar la lógica de un servidor (*cartoserver*) encargado del diálogo con el servidor de mapas y la provisión de servicios a un cliente (*cartoclient*), cuya misión es acceder mediante SOAP a los servicios proporcionados por servidores *cartoweb* y renderizar de la manera apropiada la información.

Esto permite separar lógica y físicamente los clientes (*cartoclient*) de los servidores con múltiples configuraciones (N clientes –M servidores) y posibilidades de escalado.

Funcionalmente presenta un abanico muy completo de características propias de un geoportal, con la posibilidad de ir añadiendo o desarrollando nuevos *plugins*, siendo esta otra de las fortalezas del sistema.

Permitir la representación geoespacial de la información asociada a cualquier negocio que lo requiera, también constituye uno de los objetivos fundamentales de la suscitada tecnología. De igual forma sería posible Integrar la información socioeconómica existente (recursos humanos, activos fijos, entidades de servicios, lugares de interés, etc.) con la información geográfica asociada. Proporcionándose de esta forma un conjunto de servicios para el acceso a la información geográfica, en función de su consulta, análisis y visualización, mediante una interfaz de usuario sencilla y de fácil manejo que pueda ser utilizada por usuarios no especializados en tecnología SIG.

Uno de los elementos relevantes a tener en cuenta para la construcción de este tipo de aplicaciones, es que su entorno de desaplique varía un tanto con respecto a los sistemas convencionales de gestión. Obviamente GeneSIG no es la excepción, a continuación se muestra una figura que representa gráficamente los elementos que aquí intervienen.



Nótese como se incorpora un nuevo elemento dentro del cumulo clásico de servidores a utilizar en una típica aplicación basada en el modelo cliente-servidor. Se hace evidente que el servidor de mapas es clave para este tipo de productos, teniendo en cuenta que el procesamiento en bruto de los datos cartográficos es extremadamente complejo, por tanto es mucho mas factible delegar dicha responsabilidad a un sistema externo, en acápitones posteriores se procederá a describir mejor estos elementos.

1. Conceptos

A continuación se enunciarán de forma muy breve alguno de los conceptos que se consideran claves para la comprensión de este material.

1.1 Mapa



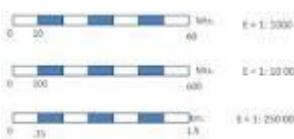
Es una representación gráfica y métrica de una porción de territorio sobre una superficie bidimensional, generalmente plana, pero que puede ser también esférica como ocurre en los globos terráqueos. El que el mapa

tenga propiedades métricas significa que ha de ser posible tomar medidas de distancia, ángulos o superficies sobre él y obtener un resultado aproximadamente exacto.

1.2 Leyenda

Explicación de los símbolos, los colores, las tramas y los sombreados empleados en un mapa; suele encontrarse a pie de página o en un recuadro, situado en sus márgenes o bien en su dorso. Los símbolos empleados en los mapas pueden llegar a contener un gran volumen de información, que por su facilidad de lectura permiten una rápida interpretación.

1.3 Escala

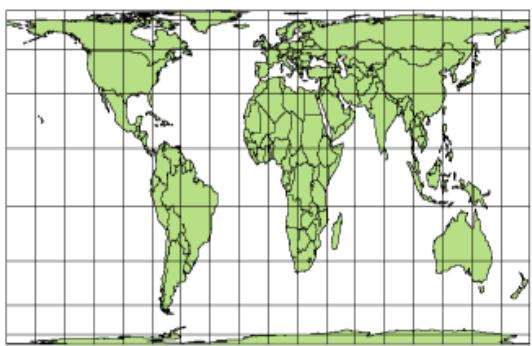


Relación entre la distancia que separa dos puntos en un mapa y la distancia real de esos dos puntos en la superficie terrestre. En los mapas, la escala puede expresarse de tres modos distintos: en forma de proporción o fracción, con una escala gráfica o con una expresión en palabras y cifras. Cuanto mayor es la escala, más se aproxima al tamaño real de los elementos de la superficie terrestre. Los mapas a pequeña escala generalmente representan grandes porciones de la Tierra y, por tanto, son menos detallados que los mapas realizados con escalas más grandes. La relación matemática entre las dimensiones en el mapa, carta o plano y la superficie terrestre que representa. Por extensión puede referirse a la mayor o menor profundidad del enfoque en un tema geográfico.

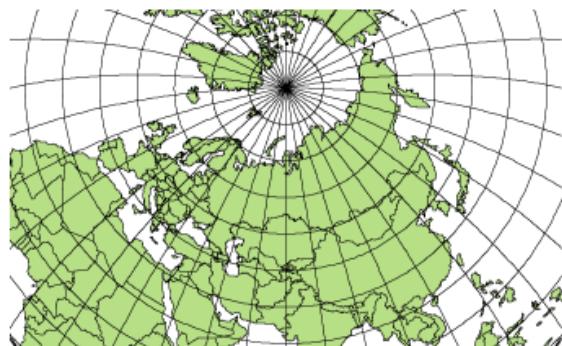
1.4 Proyección

Hoy en día, debido a la gran cantidad de usuarios de SIG, son muchas las interrogantes orientadas a manejos de proyección cartográfica que se pueden llegar a generar, pues son pocos los clientes que manejan información acerca de la correcta representación de la superficie terrestre, o que manejen cabalmente términos y soluciones a problemas de proyección, escala, huso y transformación de coordenadas, etc. Esto puede llevar a cometer errores significativos durante el ingreso de la información, aún más cuando se necesita interactuar con distintos tipos de cartografías, siendo esta una de las razones por las que se debe prestar especial atención al estudio de este tema.

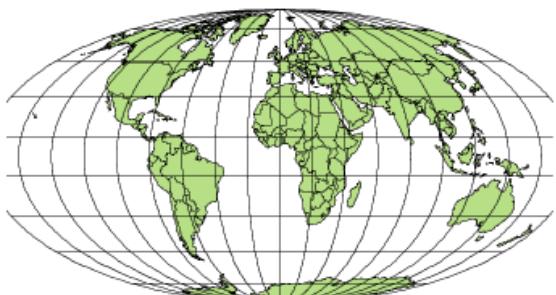
Por tanto un sistema de representación gráfico establece una relación ordenada entre los puntos de la superficie curva de la Tierra y los de una superficie plana (mapa). Estos puntos se localizan auxiliándose en una red de meridianos y paralelos, en forma de malla. La única forma de evitar las distorsiones de esta proyección sería usando un mapa esférico pero, en la mayoría de los casos, sería demasiado grande para que resultase útil.



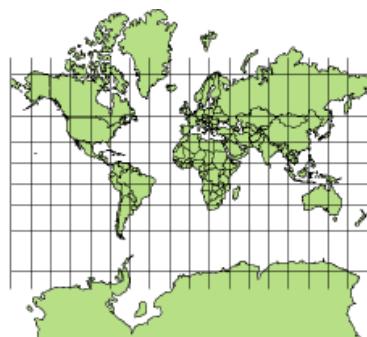
Peters-Projektion



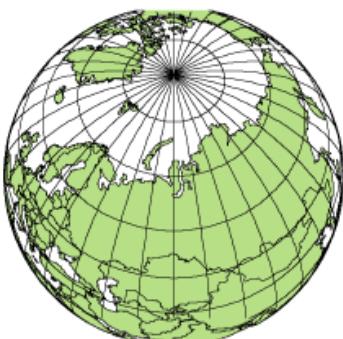
Längentreue Azimuthalprojektion



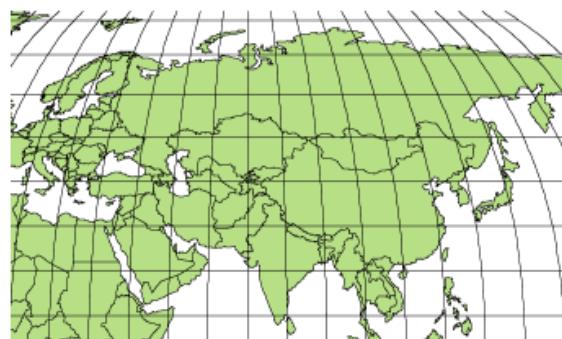
Mollweide-Projektion



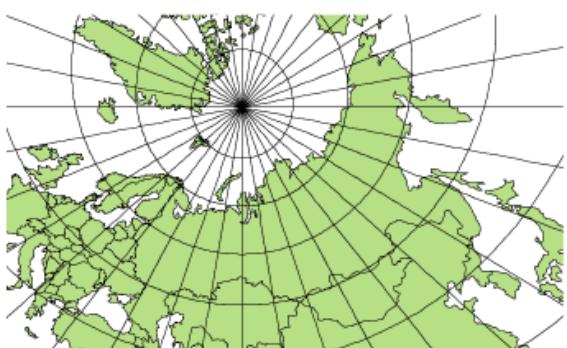
Mercator-Projektion



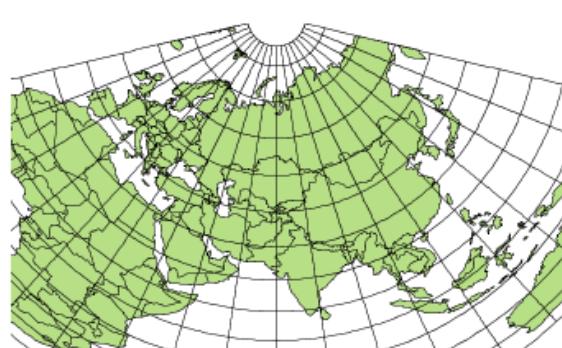
Senkrechte Umgebungsperspektive



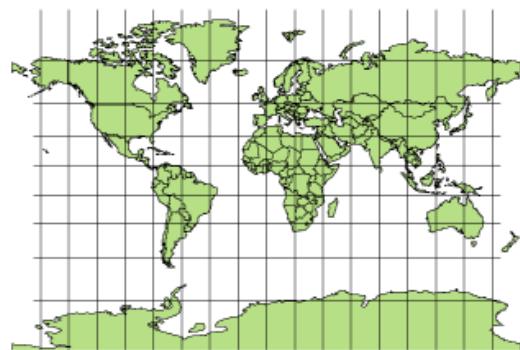
Robinson-Projektion



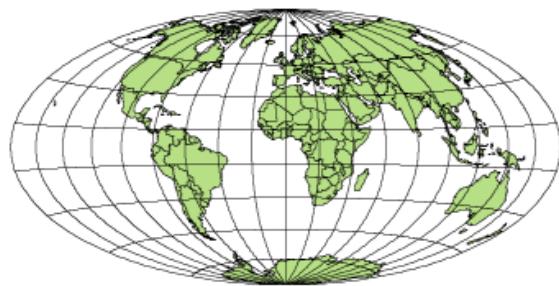
Gnomonische Projektion



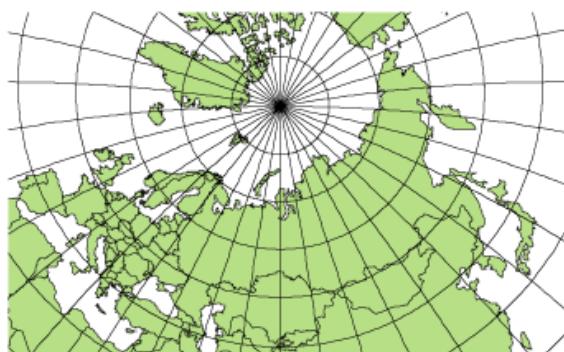
Flächentreue Kegelprojektion



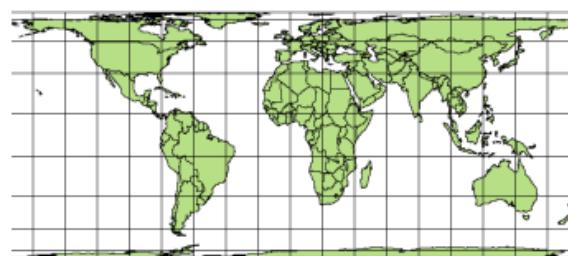
Zylinderprojektion nach Miller



Hammer-Aitoff-Projektion



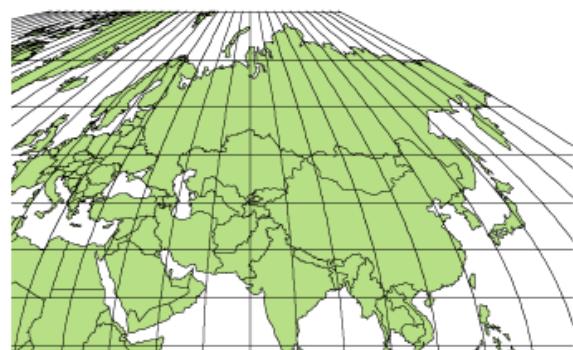
Stereographische Projektion



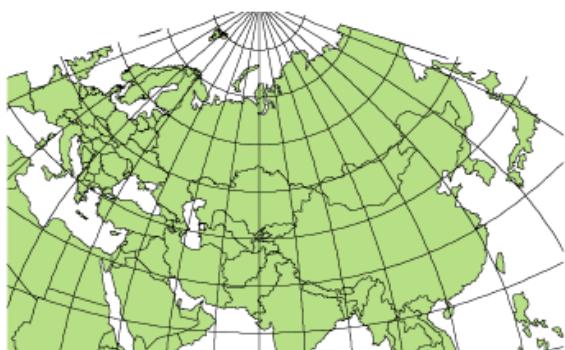
Behrmann-Projektion



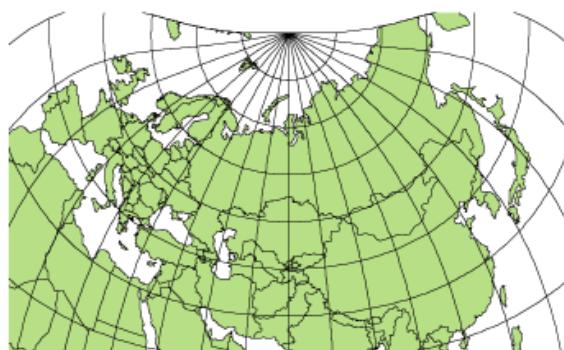
Hotine Oblique Mercator-Projektion



Sinusoidale Projektion



Transverse Mercator-Projektion



Cassini-Soldner-Projektion

Como es conocido, la principal función de la cartografía consiste en representar en un plano una parte considerable de la superficie terrestre, inclusive la totalidad de la extensión de la tierra. Tomando en cuenta que la superficie del globo terráqueo, se ha considerado como esférica con deformaciones en los polos, no es posible realizar dichas proyecciones sin deformaciones producidas por el proceso matemático. Para esto se cuenta con distintos tipos de proyecciones, los cuales poseen ciertas ventajas sobre otros sistemas de proyección. Tomando en cuenta esto, la cartografía estudia los sistemas de proyección más adecuados para definir de forma biunívoca una correspondencia matemática entre los puntos del elipsoide y sus transformaciones en el plano. Estas transformaciones van a llevar consigo una serie de deformaciones, denominadas anamorfosis, que pueden ser lineales, superficiales o angulares.

Para representar correctamente estos tipos de procesos, se deben realizar una serie de correcciones y ajustes.

- Toma de medidas en la superficie terrestre.
- Correcciones para referir estas medidas al geoide.
- Correcciones para referir los datos al elipsoide adoptado.
- Aplicación de las relaciones matemáticas propias del sistema de proyección cartográfico elegido.

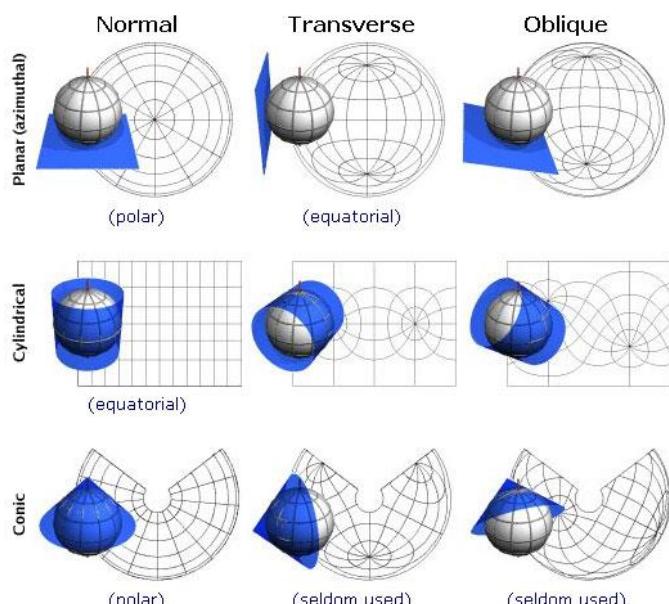


Ilustración 1 Modelos de proyecciones

A continuación se describen algunos de los sistemas de proyección mas utilizados, los cuales han sufrido diferentes evoluciones, hasta llegar a los actuales sistemas de proyección.

1.4.1 Esquema de una proyección cónica.

La proyección cónica se obtiene proyectando los elementos de la superficie esférica terrestre sobre una superficie cónica tangente, situando el vértice en el eje que une los dos polos. Aunque las formas presentadas son de los polos, los cartógrafos utilizan este tipo de proyección para ver los países y continentes.

- Proyección cónica simple
- Proyección conforme de Lambert

- Proyección cónica múltiple

1.4.2 Esquema de una proyección azimutal, cenital o polar

En este caso se proyecta una porción de la Tierra sobre un plano tangente al globo en un punto seleccionado, obteniéndose una imagen similar a la visión de la Tierra desde un punto interior o exterior. Si la proyección es del primer tipo se llama proyección gnomónica; si es del segundo, ortográfica. Estas proyecciones ofrecen una mayor distorsión cuanto mayor sea la distancia al punto tangencial de la esfera y el plano. Este tipo de proyección se relaciona principalmente con los polos y hemisferios.

- Proyección ortográfica
- Proyección estereográfica
- Proyección gnomónica
- Proyección azimutal de Lambert

1.4.3 Esquema de una proyección cilíndrica

Consideran la superficie del mapa como un cilindro, que rodea al globo terráqueo tocándolo en el ecuador, mientras que los meridianos y paralelos son líneas rectas que se cortan perpendicularmente entre sí.

En distintos tipos de proyecciones cilíndricas se distingue que, debido a la curvatura de la tierra, los paralelos de latitud más cercanos a los polos aparecen cada vez más próximos entre sí. El mapa resultante representa la superficie del mundo como un rectángulo con líneas paralelas equidistantes de longitud y líneas paralelas de latitud con separación desigual. Debido a que las áreas se van distorsionando a medida que se acercan a los polos, este tipo de proyecciones no son recomendables para zona de latitudes mayores al los 40 grados de latitud. Entre las más utilizadas se encuentra Mercator y de Peters.

La popular proyección de Mercator, desarrollada por el geógrafo flamenco Gerardus Mercator, es una proyección cilíndrica y a la vez conforme. Un mapa con este tipo de proyección es muy exacto en las regiones cercanas al Ecuador, pero va perdiendo calidad ya que se distorsiona bastante en las áreas de latitudes altas. Todas las direcciones se presentan con gran precisión, no así todas sus áreas y distancias.

1.4.4 La proyección Universal transversa de Mercator

Es una proyección cilíndrica transversal secante. Basada en la proyección de Mercator, es aplicable principalmente a grandes zonas de la superficie terrestres, para ello se emplean distintos cilindros tangentes a varios meridianos. Debido a que este sistema es aplicado a grandes extensiones de longitud a medida que comienza a alejarse del meridiano central, causará deformaciones considerables, debido a esto se acude a subdividir la superficie terrestre en 60 husos o zonas de igual cantidad de grados, en cuanto a su longitud, es decir 3 grados a cada lado de su meridiano central, por lo tanto quedara conformado por franjas de 6 grados de extensión.

En la proyección U.T.M. tanto los paralelos como los meridianos son líneas curvas. Es fácil distinguir que si proyectamos el mundo en su totalidad y a medida que nos alejamos de la línea del Ecuador, la distancia entre los paralelos aumenta considerablemente,

consecuente de esto la proyección realizada en la cercanía de los polos posee una deformación mayor. Por esta razón la UGGI (Unión Geodésica y Geofísica Internacional), ha dictaminado que todos los husos debían acotar su latitud a 84 grados, ya sea en el Norte como en el Sur. Para reducir lo más posible la distorsión, cada zona sólo tiene 6° de longitud, es por esto que la distorsión en los extremos (3° a cada lado) es bastante aceptable. Cada país tiene uno o más de estas zonas que cubren su territorio. Por ejemplo, las zonas correspondientes a Chile Continental son la zona 18 y 19.

1.4.4 Sistemas de referencia utilizados en ambientes SIG

Antiguamente, cada elipsoide se ubicaba en diferentes posiciones o puntos de referencia (datum), debido a esto, sólo se obtenía un buen ajuste para el área de trabajo. Sin embargo con el desarrollo de fórmulas matemáticas se ha permitido relacionar la diferencia de posición del centro de estos elipsoides con respecto al WGS84 (puesto que éste es un elipsoide geocéntrico). Dicho trabajo ha logrado aceptables niveles de precisión, el cual fue desarrollado por el científico Ruso Sergui Molodensky. Su labor consistió en determinar la variación en metros (x, y, z) en cuanto a la distancia de desfase del centro de los respectivos elipsoides. Estos valores son los que se usan en casi todos los software de SIG. Los cuales se visualizan en la tabla. 2.1, que se expondrá a continuación:

Constantes de Molodensky				
Datum	Elipsoide	delta X	delta Y	delta Z
PSAD56 Promedio	Internacional de 1909 / 1924 (Hayford)	-288	175	-376
PSAD56 Norte Chile		-270	183	-390
PSAD56 Sur Chile		-305	243	-442
SAD69 Promedio	Internacional 1969	-57	1	-41
SAD69 Chile		-75	-1	-44

Ilustración 2 Constantes de transformación de SR¹, según ecuaciones de Molodensky.

En general, un elipsoide puede ser usado en la definición de una gran cantidad de datum, en el caso opuesto no ocurre lo mismo, pues un datum tiene asociado sólo un elipsoide.

El WGS84, también es un sistema convencional terrestre, donde el origen de coordenadas X, Y, Z está en el centro de masa de la tierra. El eje Z pasa por el polo convencional terrestre, el eje X es la intersección entre el meridiano y el origen de las longitudes, y el eje Y completa una terna de ejes fijos a la tierra en forma perpendicular a la anterior. De ésta forma la terna definida sirve de centro geométrico del elipsoide WGS84 y su eje Z es su eje de revolución. Éste elipsoide de revolución asignado al sistema cartesiano antes mencionado, posee los parámetros correspondientes al Sistema de Referencia 1980 (GRS80). Estos son los siguientes:

¹ SR acrónimo de Sistemas de Referencia

Semieje mayor	a	6378137 m
Achatamiento	f	1/ 298.257223563
Velocidad Angular de la Tierra	ω	7292115 E -11 rad/s
Constante Gravitacional	μ	3986004.418 E +8 m ³ /s ²

Ilustración 3 Parámetros geodésicos asignados a WGS84

Tomando en cuenta que se han refrescado la gran mayoría de los conceptos técnicos empleados en cuanto a la parte cartográfica, en acápite posteriores se procederá a llevar a cabo la vinculación con MapServer, el cual maneja todo estos tipos de procesos a través de la librería *proj4*.

1.5 Capa

Conjunto lógico de elementos temáticos descritos y almacenados en una biblioteca. Las *Layers* o capas organizan la biblioteca según temas.

1.6 Modelo de datos Raster

Un tipo de datos raster es, en esencia, cualquier tipo de imagen digital representada en mallas. El modelo de SIG raster o de retícula se centra en las propiedades del espacio más que en la precisión de la localización. Divide el espacio en celdas regulares donde cada una de ellas representa un único valor.

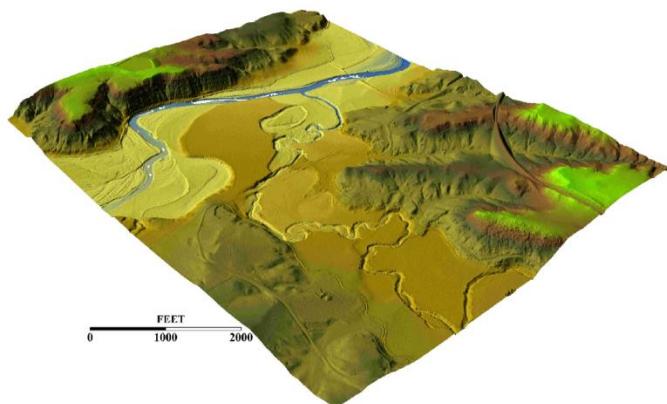


Ilustración 4 Ejemplo del ejemplo del raster para describir el terreno

1.7 Modelo de datos Vectoriales

En los datos vectoriales, el interés de las representaciones se centra en la precisión de localización de los elementos geográficos sobre el espacio y donde los fenómenos a representar son discretos, es decir, de límites definidos. Cada una de estas geometrías está vinculada a una fila en una base de datos que describe sus atributos.

1.8 Puntos

Los puntos se utilizan para las entidades geográficas que mejor pueden ser expresadas por un único punto de referencia. En otras palabras: la simple ubicación. Los puntos

transmiten la menor cantidad de información de estos tipos de archivos y no son posibles las mediciones. También se pueden utilizar para representar zonas a una escala pequeña.

1.9 Líneas

Las líneas unidimensionales o polilíneas son usadas para rasgos lineales como ríos, caminos, ferrocarriles, rastros, líneas topográficas o curvas de nivel. De igual forma que en las entidades puntuales, en pequeñas escalas pueden ser utilizados para representar polígonos. En los elementos lineales puede medirse la distancia.

1.10 Poligonales

Los polígonos bidimensionales se utilizan para representar elementos geográficos que cubren un área particular de la superficie de la tierra. Estas entidades pueden representar lagos, límites de parques naturales, edificios, provincias, o los usos del suelo, por ejemplo. Los polígonos transmiten la mayor cantidad de información en archivos con datos vectoriales y en ellos se pueden medir el perímetro y el área.

1.11 Información socioeconómica

Es un conjunto organizado de datos procesados referentes al aspecto social y económico de cualquier lugar de interés del país.

1.12 Modelos Digitales del Terreno

Uno de los elementos básicos de cualquier representación digital de la superficie terrestre son los Modelos Digitales de Terreno. Constituyen la base para un gran número de aplicaciones en ciencias de la Tierra, ambientales e ingenierías de diverso tipo. Pueden ser construidos directamente a partir de la realidad a representar, pero es muy habitual que exista un modelo analógico intermedio a partir del cual se realiza la codificación.

En su definición existen dos condiciones suplementarias. Debe existir una estructura interna que represente las relaciones espaciales entre los datos y la variable representada en el modelo debe ser cuantitativa y de distribución continua. Se puede definir un MDT como una estructura numérica de datos que representa la distribución espacial de una variable cuantitativa y continua. [6]

Es la representación simplificada de la topografía del terreno. Para ello se considera que las elevaciones forman una superficie tridimensional ondulada, en la que dos dimensiones se refieren a los ejes de un espacio octogonal plano (X , Y), y la tercera mide la "altura" (Z). (Figura 4) Por ello, se suele hablar de representaciones gráficas con dos dimensiones topológicas y media. [5]

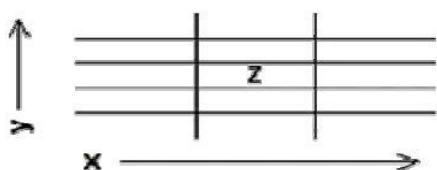


Ilustración 5 Sistema de Referencia Espacial.

Son modelos simbólicos, ya que las relaciones de correspondencia que se establecen con el objeto real tienen la forma de algoritmos o formalismos matemáticos. Forman una estructura de datos, lo que significa que no son sólo una acumulación o lista de cifras, sino que guardan relaciones entre ellos.

El trabajo con un MDT incluye las siguientes fases que no son necesariamente consecutivas en el tiempo:

- Generación del MDE.
- Manipulación del MDE para obtener otras capas del MDT.
- Análisis del MDT.
- Visualización en dos dimensiones o mediante levantamientos 3D de todas las capas para localizar errores.
- Aplicación del MDT.

Los MDT presentan algunas ventajas sobre los diferentes tipos de modelos existentes:

- **Repetibilidad:** Los resultados no se someten a factores aleatorios o incontrolados a menos que estén expresamente diseñados. Así como pueden estar comprobados y replicados las veces que se desee.
- **No ambigüedad:** Cada elemento del modelo tiene propiedades, valores específicos y explícitos.
- **Verificabilidad:** Los resultados pueden ser analizados uno a uno y comprobados en todas las fases. Estos se construyen mediante pasos explícitos y concretos.

1.13 Modelo Digital de Elevaciones

Un Modelo Digital de Elevaciones se define como una estructura numérica de datos que representa la distribución espacial de la altitud de la superficie del terreno. Consiste en una serie de puntos con coordenadas conocidas o referenciadas a un sistema de coordenadas bidimensionales a las que se les asocia un valor de elevación.

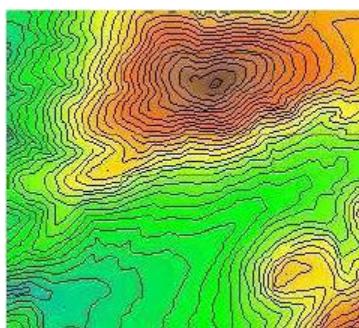


Ilustración 6 Modelo Digital de Elevación.

Un terreno puede describirse de forma genérica como una función bivariable continua $z = Z(x, y)$ donde z representa la altitud del terreno en el punto de coordenadas (x, y) y Z es una función que relaciona la variable con su localización geográfica. La función anterior se aplica sobre un dominio espacial concreto D . En consecuencia, un MDE puede describirse genéricamente como $MDE = (D, Z)$. [7]

Por este motivo, se han ensayado numerosas opciones en la búsqueda de una forma de representar y almacenar la altitud, donde se equilibre la pérdida de información y algunos efectos secundarios indeseables, como son el excesivo tamaño de los archivos o la dificultad de manejo. [7]

Mientras que los mapas convencionales usan casi exclusivamente una única convención (las curvas de nivel) para la representación de la superficie del terreno, los MDE disponen de alternativas más variadas, desde una transposición casi directa de las isohipsas hasta otras menos habituales en la cartografía impresa pero más adaptada al proceso digital. [6]

Los valores de elevación pueden ser manipulados digitalmente y desplegados en un monitor como una malla o como un conjunto de celdas, a la que se asocian los valores de altura a cada una de las intersecciones de líneas de la malla. Para este caso, la presentación visual es una vista isométrica. (Figura 6)

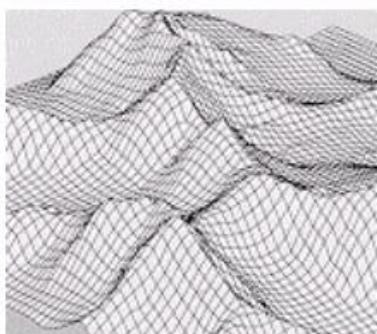


Ilustración 7 Representación tipo malla en vista isométrica

Para el caso de que a las celdas de una cuadrícula ráster se le asignen los valores correspondientes a los intervalos de alturas diferenciados por gamas, ya sea de tonos de gris o de colores, la presentación gráfica puede ser en dos o tres dimensiones.

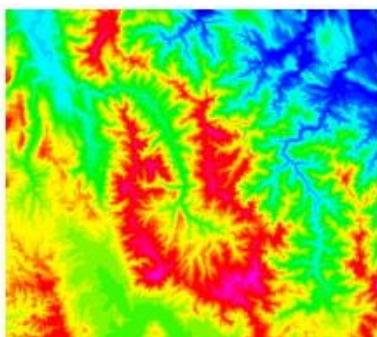


Ilustración 8 Representación en dos dimensiones por medio de tonos

1.14 Open Geospatial Consortium

El OGC² fue creado en 1994 y agrupa a una gran cantidad de organizaciones públicas y privadas. Su creación se produjo con el fin de obtener la definición de estándares abiertos que permitan procesar información de diversos lugares del mundo en conjunto, dentro de los Sistemas de Información Geográfica. Esto con la factibilidad de obtener acuerdos entre las diferentes empresas del sector, que posibiliten la interoperabilidad de sus

² OGC acrónimo de Open Geospatial Consortium

sistemas de geoprocесamiento y así facilitar el intercambio de información geográfica en beneficio de los usuarios. Anteriormente fue conocido como Open GIS Consortium.

Las especificaciones más importantes surgidas del OGC son:

- **GML**: Lenguaje de Marcado Geográfico
- **WFS**: Web Feature Service
- **WMSP**: Web Map Service
- **WCS**: Web Coverage Service
- **CS-W**: Catalog Service Web

Los productos y los servicios que se conforman a las especificaciones de interfaz abiertas de OGC permiten a usuarios intercambiar y aplicar libremente la información espacial, usos y servicios a través de redes, diversas plataformas y productos

2. Elementos técnicos necesarios

A continuación se enunciarán algunos elementos técnicos a consideración del equipo de trabajo de GeneSIG que son indispensables en función de desarrollar sobre dicha plataforma.

2.1 SIG

Sistema de Información Geográfica: Es el “conjunto de métodos, herramientas y datos que están diseñados para actuar coordinada y lógicamente para capturar, almacenar, analizar, transformar y presentar toda la información geográfica y de sus atributos con el fin de satisfacer múltiples propósitos. Los SIG son una tecnología que permite gestionar y analizar la información espacial, y que surgió como resultado de la necesidad de disponer rápidamente de información para resolver problemas y contestar a preguntas de modo inmediato”. (*RODRÍGUEZ et al., 1998*).

2.2 OOP

La programación orientada a objetos OOP³ no va a ser explicada en este capítulo, ya que se necesitaría un libro entero para ello. Como GeneSIG hace un uso continuo de los mecanismos orientados a objetos disponibles en PHP 5, es un requisito obligatorio el conocer la OOP antes de aprender GeneSIG.

De forma muy sintetizada se puede afirmar que, la idea de la programación orientada a objetos es que una aplicación se puede considerar como una colección de unidades individuales, llamadas objetos, que interactúan entre sí. Los programas tradicionales pueden considerarse como una colección de funciones o como una lista de instrucciones de programación.

PHP 5 incluye los conceptos de clase, objeto, método, herencia y muchos otros propios de la programación orientada a objetos. Aquellos que no estén familiarizados con estos conceptos, deberían consultar la documentación oficial de PHP disponible en <http://www.php.net/manual/es/language.oop5.basic.php>.

³ OOP acrónimo de Object Oriented programming

2.3 PHP

GeneSIG está programado en PHP 5 y está enfocado al desarrollo de aplicaciones web en el mismo lenguaje de programación. Por este motivo, es obligatorio disponer de unos conocimientos avanzados de PHP 5 para sacar el máximo partido al *framework*. La versión mínima de PHP requerida para ejecutar GeneSIG es PHP 5.2. (*The PHP Group, 2001*)

2.4 JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador aunque existen implementaciones de dicho lenguaje del lado del servidor conocidos como SSJS⁴.

2.5 PEAR

PEAR⁵ es un "framework y sistema de distribución para componentes PHP reutilizables". PEAR permite descargar, instalar, actualizar y desinstalar scripts de PHP. Si se utiliza un paquete de PEAR, no es necesario decidir donde guardar los scripts, cómo hacer que se puedan utilizar o cómo extender la línea de comandos (CLI). PEAR es un proyecto creado por la comunidad de usuarios de PHP, está desarrollado con PHP y se incluye en las distribuciones estándar de PHP.

Sugerencia El sitio web de PEAR, <http://pear.php.net/>, incluye documentación y muchos paquetes agrupados en categorías. PEAR es el método más profesional para instalar librerías externas en PHP. GeneSIG aconseja el uso de PEAR para disponer de una instalación única y centralizada que pueda ser utilizada en varios proyectos. (*The PHP Group, 2001*)

2.6 Ajax

El Ajax⁶ es una técnica de desarrollo web para crear aplicaciones interactivas o RIA⁷. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

⁴ SSJS acrónimo de Server-side JavaScript

⁵ PEAR acrónimo de PHP Extension and Application Repository

⁶ Ajax acrónimo de Asynchronous JavaScript And XML

⁷ RIA acrónimo de Rich Internet Applications

2.7 CGI

Interfaz de entrada común es una importante tecnología de la World Wide Web que permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web. CGI⁸ especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME⁹. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGI.

Las aplicaciones CGI fueron una de las primeras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este programa puede estar escrito en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

2.8 Servidor Web

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales, así como síncronas o asíncronas con el cliente generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se utiliza el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa.

2.9 Servidor de Mapas

Un servidor de cartografía digital también conocido IMS¹⁰, provee cartografía a través de la red tanto en modo vectorial como con imágenes. La especificación estándar para estos servidores es la OGC¹¹ Web Map Service. Puede utilizarse para crear servicios de información basados en mapas dinámicos, imágenes satelitales, datos SIG o un conjunto de estos. Permite ejecutar una visualización sobre cualquier mapa publicado que este accesible desde Internet, además de poder realizar funciones sobre la cartografía como por ejemplo, acercamientos, desplazamientos, localización de sitios en el mapa, y la obtención de alguna característica espacial. El usuario puede acceder a la información desde su Navegador o Browser habitual sin la necesidad de contar con algún programa en especial, solo necesitando una conexión a Internet.

En la actualidad existe una variada oferta de herramientas de este tipo tanto de licencia comercial como libre, algunos de los más usados son:

⁸ CGI acrónimo de Common Gateway Interface

⁹ MIME acrónimo de Multipurpose Internet Mail Extensions

¹⁰ IMS acrónimo de Internet Map Server

¹¹ OGC acrónimo de Open Geospatial Consortium

- MapObjects IMS
- ArcIMS
- MapGuide
- MapServer
- GeoTools
- GIS Viewer
- Spatial Direct

La posibilidad de llevar a cabo la publicación de mapas en Internet es un tema que es necesario discutir ya que provee desde el punto de vista de la infraestructura nacional de información geográfica un acceso a datos por parte de un gran numero de usuarios, con lo que su implementación y desarrollo en forma masiva daría como resultado una mejor integración de la información geográfica para dar solución a problemas de variada índole. El interés de los usuarios en disponer de información geográfica en la red junto con la capacidad de obtener y representar información de diferentes fuentes, ha conseguido que el mundo empresarial este realizando un gran esfuerzo para cubrir estas necesidades.

Algunas de las ventajas que nos entrega un servidor de mapas son:

- Acceso a información espacial por parte de un gran numero de personas.
- Requerimientos menores de Software.
- Constante y rápida evolución de tecnologías aplicada a esta área.
- Facilidad de análisis de datos y atributos espaciales.

	MapServer 5.2.1	GeoServer 1.7.3	MapGuide 2.02	ArcIMS 9.2
Licenciamiento	MapServer se distribuye con la licencia de la Universidad de Minesota, la cual le da libertad al usuario de utilizar, copiar y distribuir sin ninguna limitación, el software no incluye garantía.	GeoServer se distribuye con la licencia GPL o General Public License.	MapGuide es distribuida con la licencia LGPL o Lesser General Public License	ArcIMS es un marca registrada de ESRI, el código es cerrado y su utilización se la realiza previa la obtención de una licencia comercial.
Sistemas Operativos soportados	Windows, Linux Mac OS X Código Fuente en C.	Windows, Mac OS X Binarios(Plataforma independiente) Código Fuente.	Windows, Código Fuente en C requiere FDO 3.3.1	Windows Linux Sun Solaris HP-UX IBM AIX
Tecnología	CGI, FastCGI	Spring J2EE	FastCGI, ISAPI, SWIG	Java J2EE
Servidores de Aplicación soportados	No	Tomcat Jetty Weblogic JBoss Cualquier contenedor de Servlets.	No	Tomcat 5.5.17 / 6.0.13 Servlet Exec 5.0 / 6.0 ISAPI JBoss 4.0.2 Oracle Application Server Sun Java Application Server 9.1 WebLogic 9.2 / 10
Servidores Web Soportados	Apache IIS	Apache IIS	Apache IIS	Apache 2.2 .4 IIS Sun Java System Web Server 7

Ilustración 9 Elementos de comparación entre servidores de mapas

Las ventajas mencionadas anteriormente nos dan la posibilidad de insertar este tipo de herramientas en el que hacer de la vida cotidiana, como un instrumento informativo que ayude en el proceso de toma de decisiones y un mejor aprovechamiento de la información disponible, pero de difícil acceso.

- **MapServer:** <http://mapserver.gis.umn.edu/dload.html>
- **GD:** <http://www.boutell.com/gd>
- **FreeType:** <http://www.freetype.org>
- **libJPEG:** <http://www.ijg.org/files>
- **libpng:** <http://www.libpng.org/pub/png>
- **zlib:** <http://www.gzip.org/zlib>
- **GDAL:** <http://gdal.maptools.org>
- **Proj.4:** <http://proj.maptools.org>
- **Shapelib:** <http://shapelib.maptools.org>

2.10 Servidor de Bases de Datos

Los servidores de bases de datos surgen con motivo de la necesidad de las empresas de manejar grandes y complejos volúmenes de datos, al tiempo que requieren compartir la información con un conjunto de clientes de una manera segura. Ante este enfoque, un SGBD¹² deberá ofrecer soluciones de forma fiable, rentable y de alto rendimiento el cual no es más que un sistema bajo arquitectura *cliente/servidor* que proporciona servicios de gestión, administración y protección de la información a través de conexiones de red, gobernadas por unos protocolos definidos y a los que acceden los usuarios, de modo concurrente, a través de aplicaciones clientes.

3. Procedimiento de instalación

Este acápite está dedicado a describir los elementos necesarios que deben ser instalados en función de garantizar el correcto funcionamiento un sistema de información geográfico, cuyo desarrollo se sustente sobre la base tecnológica definida por la plataforma GeneSIG.

3.1 Servidor Web

3.1.2 Virtualización

El término *Virtual Hosting* se refiere a la práctica de publicar más de un sitio web en una sola máquina. En otras palabras, no es más que un método para albergar múltiples nombres de dominio en un único nodo de procesamiento o conjunto de estos. Permitiendo que un servidor pueda compartir sus recursos, como la memoria y ciclos de procesador. Sin necesidad de que todos los servicios prestados tengan que utilizar el mismo nombre de host

Este recurso es realmente importante pues garantiza principalmente en tiempo de desarrollo evitar la duplicidad del código fuente asociado a nuestros proyectos. Permitiendo tener publicado los mismos sin importar si quiera que estos se encuentren en distintas particiones. Siendo esto también un elemento primordial en cuestiones de seguridad, pues se estarían publicando solo aquellos directorios destinados para tal objetivo. Evitándose de esta forma la práctica irresponsable que comúnmente utilizan la mayoría de los desarrolladores, al copiar todo el producto en el directorio de publicación del Apache.

Otro elemento interesante es que los servidores virtuales se clasifican principalmente en dos grandes grupos, los cuales se basan en:

- **IP:** Utiliza una dirección *ip* independiente para cada nombre de host, y se puede realizar con cualquier protocolo.
- **Nombres:**

Esto guarda las direcciones IP y la sobrecarga administrativa asociada pero el protocolo

¹² **SGBD** acrónimo de Sistema Gestor de Bases de Datos

que se sirve debe proporcionar el nombre de host en un momento adecuado. En particular, existen dificultades significativas con nombre basado hosting virtual con SSL / TLS

A continuación se muestra un ejemplo que ilustra como crear un *virtual host* denominado *gis.os* alojado en el directorio */media/datos/work/web* y que se mantendrá a la escucha por el puerto 5.

El primer paso consiste en copiar el contenido definido por el fichero *default* ubicado en */etc/apache2/sites-available/*, en función de tomarlo como plantilla para generar un archivo denominado *gis.os* en el mismo directorio.

```
<VirtualHost *:5>
    ServerName gis.os
    ServerAdmin webmaster@localhost
    DocumentRoot /media/datos/work/web

    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>

    <Directory /media/Datos/Work/web/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow, deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/

    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Esto mismo se pudo realizar sobre el fichero *default*, sin embargo no se recomienda dicha práctica para evitar posibles errores o incompatibilidades de configuración. Observese como se especifica en la etiqueta denominada *VirtualHost* la opción **:5*, esto indica que para cualquier dirección ip que coincida con el puerto 5 se aplica dicha regla. Otra de las opciones más relevantes son *ServerName* y *DocumentRoot*, donde la primera se refiere al nombre que define el directorio de publicación y la segunda a la dirección física que contendrá los archivos.

Por tanto se ejecuta la utilidad *ls* sobre el directorio definido para los sitios habilitados sobre apache arrojaría una salida similar a la que se muestra a continuación:

```
root@pc-sig11:/home/me# ls /etc/apache2/sites-available/ -all
drwxr-xr-x 2 root root 4096 sep 14 09:28 .
drwxr-xr-x 7 root root 4096 sep 7 10:56 ..
-rw-r--r-- 1 root root 693 sep 14 09:27 default
```

```
-rw-r--r-- 1 root root 7251 abr 1 2012 default-ssl  
-rw-r--r-- 1 root root 936 sep 14 11:20 gis.os
```

El Segundo paso consiste en crear un enlace simbólico del fichero anterior en el directorio definido para los sitios habilitados del *apache*, permitiendo que el mismo cargue nuestra configuración. Para ello se procede a redireccionar la consola, en función de generar dicho *link* a través de la utilidad denominada *ln*.

```
root@pc-sig11:/home/me# cd /etc/apache2/sites-enabled/  
root@pc-sig11:/home/me# ln -s ..sites-available/gis.os
```

Nótese la importancia de estar previamente ubicado en el directorio */etc/apache2/sites-enabled/*, de lo contrario la referencia al enlace se perdería pues esta se especifica a partir de una ruta relativa. En caso de listarse la información contenida en dicho directorio se mostraría una salida similar a la que aparece a continuación:

```
root@pc-sig11:/home/me# ls /etc/apache2/sites-enabled/ -all  
drwxr-xr-x 2 root root 4096 sep 14 09:36 .  
drwxr-xr-x 7 root root 4096 sep 7 10:56 ..  
lrwxrwxrwx 1 root root 26 sep 7 10:56 default -> ..sites-available/default  
lrwxrwxrwx 1 root root 25 sep 14 09:36 gis.os -> ..sites-available/gis.os
```

Luego se procede a especificar el nombre definido para el *host virtual* en el fichero denominado *hosts* ubicado en el directorio */etc/*. Especificado el par que permite asociar la dirección ip con el alias definido, tal y como se puestra a continuación:

127.0.0.1	localhost
127.0.0.1	gis.os

Por ultimo se debe especificar el puerto asociado a la regla previamente definida en el archivo */sites-available/gis.os*, en el fichero denominado *ports.conf* ubicado en el directorio */etc/apache2/*, tal y como se muestra a continuación:

```
NameVirtualHost *:5  
Listen 5
```

pueden ser "basada en IP", lo que significa que usted tiene una dirección IP diferente para cada sistema, o "basado en nombres", lo que significa que tiene varios nombres que se ejecutan en cada dirección IP. El hecho de que se están ejecutando en el mismo servidor físico no es aparente para el usuario final.

Apache fue uno de los primeros servidores para soportar IP hosts virtuales basados en nada más sacarlo de la caja. Las versiones 1.1 y posteriores de Apache apoyar tanto a los hosts virtuales basados en IP y el nombre de base (vhosts). La última variante de máquinas virtuales es a veces también llamado host-based o no IP hosts virtuales.

Una aplicación ampliamente utilizada es compartido alojamiento web. Precios de alojamiento web compartido son inferiores a un servidor web dedicado, ya que muchos clientes pueden ser alojados en un único servidor. También es muy común que una sola entidad a querer usar varios nombres en la misma máquina para que los nombres pueden reflejar servicios que se ofrecen en lugar de que estas prestaciones pasan a ser acogido.

Hay dos tipos principales de *virtual hosting*, el nombre de base y basadas en IP. Nombre usa hosting virtual basado en el nombre de host presentada por el cliente.. Por otra parte la técnica basada en IP utiliza una dirección IP independiente para cada nombre de host, y se puede realizar con cualquier protocolo, sino que requiere una dirección IP dedicada por cada nombre de dominio servían. Puerto hosting virtual basado también es posible en

principio, pero rara vez se utiliza en la práctica, ya que es poco amistoso para los usuarios.

Nombre basados en IP y hosting virtual basado pueden combinar, un servidor puede tener varias direcciones IP y servir a varios nombres en alguna de esas direcciones IP. Esta técnica puede ser útil cuando se utiliza SSL / TLS con certificados comodín. Por ejemplo, si un operador tiene dos certificados de servidor para un *. Example.com y otro para *. Example.net podía servir foo.example.com bar.example.com y fuera de la misma dirección IP, pero se necesita una dirección IP independiente para baz.example.net.

3.1.3 Privilegios de ejecución

En no pocas ocasiones nos enfrentamos al desarrollo de sistemas que requieren en el servidor ciertos procesamientos, los cuales tienen como peculiaridad que su ejecución corresponde a privilegios administrativos. Siendo esto una limitante establecida para los servidores de aplicaciones web, sustentada sobre la base de evitar a toda costa huecos de seguridad.

Una de las alternativas utilizadas principalmente sobre sistemas GNU/Linux, se basan en la modificación de los permisos asociados al usuario de ejecución de *Apache*, generalmente denominado *www-data*. Para lograr dicho objetivo se procede a incorporar en el fichero */etc/sudoers* una línea que contenga *www-data ALL=(root) NOPASSWD:ALL*. Esto garantiza que el usuario *www-data* se ejecute como súper usuario para la ejecución de cualquier comando, sin pedirle confirmación de clave y desde cualquier dirección *IP*.

Aplicando el procedimiento anterior se genera un hueco de seguridad, que le permite al servidor apache mediante funciones nativas del propio lenguaje *script* que se decida utilizar, ejecutar aplicaciones externas con privilegios de super usuario. Para ello se debe especificar el prefijo denominado *sudo*. Por ejemplo en caso que se desee reiniciar el propio servidor apache desde php sería de la siguiente forma:

```
exec('sudo service apache2 restart');
```

Nótese que se emplea el recurso *exec* en función de ejecutar la aplicación denominada *service*, de igual forma se puede utilizar *popen*, *system*, *passthru*, *pcntl_exec*, entre otros. Sin embargo existen otras funciones nativas del lenguaje que carecen de los privilegios para ejecutarse tales como *rmdir*, *mkdir*, *file_get_contents*, *file_put_contents*, etc. En función de solventar estas limitantes se muestra a continuación el contenido de un fichero escrito en *bash* que permite principalmente la manipulación de fichero.

```
#!/bin/bash
##
# description: Script que permite asignarle privilegios root al apache, así como la gestión básica de ficheros
# author: TON3 GIS
# date: 02/10/2012
# version: 1.0
##
if [ $# -gt 0 ]
then
    case $1 in
        chmod)
            if [ $2 -n ]
                then usr="www-data"
                else usr=$2
            fi
            if [ $3 -n ]
                then path="/etc/sudoers"
            fi
    esac
fi
```

```

        else path=$3
    fi

    msg1="msg: process to install is loading ..."
    msg2="msg: process to install is finishing ..."

    den="NO"
    host="ALL"
    runas="(root)"
    cmnd="ALL"
    opt="PASSWD"
    echo $msg1
    echo $usr $host=$runas $den$opt:$cmnd >> $path
    echo $msg2

    ;;
copy) echo $2 > $3 ;;
add) echo $2 >> $3 ;;
del)
    if [ $4 -n ]
        then path=$3
        else path=$4
    fi

    tr -d $2 < $3 >> tmp$path
    cat tmp$path > $path
    rm tmp$path
    ;;
replace)
    if [ $5 -n ]
        then path=$4
        else path=$5
    fi

    tr "$2" "$3" < $4 >> tmp$path
    cat tmp$path > $path
    rm tmp$path
    ;;
*)
    echo "msg: No comand"
    ;;
esac
else
    echo "Comand list to select: chmod|copy|add|del|replace";
fi

```

Este *script* se compone por varias utilidades, las cuales se describen a continuación:

- **chmod:** Permite habilitar los permisos de administración para el usuario de ejecución del servidor de aplicaciones web. Recibe dos parámetros, el primero corresponde al nombre de dicho usuario, tomando por defecto valor *www-data* y el segundo hace referencia a la dirección de fichero que generalmente se encuentra en el directorio */etc/* denominado *sudoers*. A continuación se muestran algunos ejemplos.

```
exec("cmd.sh chmod");
```

Nótese que el script se encuentra en la raíz del proyecto y se denomina *cmd.sh*. En caso que se requiera especificar para otro usuario nombrado *userapp* sería:

```
exec("cmd.sh chmod userapp ");
```

En caso de encontrarse el fichero necesario en el directorio */etc/root/sudoers* quedaría de la siguiente forma:

```
exec("cmd.sh chmod userapp /etc/root/sudoers");
```

- **copy:** Permite copiar una informacion determinada en un archivo específico, en

caso de no existir este lo crea, de lo contrario lo sobrescribe. Recibe dos parámetros, el primero se refiere al contenido que será insertado y el segundo define la ruta de acceso al fichero que será modificado o creado en función de su estado anterior.

```
exec("cmd.sh copy 'regex in box 786-549' /home/me/register.log");
```

- **add:** Este recurso es de comportamiento similar al anterior, la diferencia radica en que el contenido se adiciona al final del fichero sin eliminar la información previamente almacenada.

```
exec("cmd.sh add 'regex in box 786-549' /home/me/register.log");
```

- **del:** Permite eliminar un texto determinado en un fichero especificado. Recibe tres parámetros, el primero determina la información a borrar, el segundo la url en que se encuentra el fichero en cuestión y el tercero en caso de ser especificado genera un nuevo archivo con los cambios realizados sin afectar el original, siendo de carácter opcional.

```
exec("cmd.sh del 'regex' /home/me/register.log");
```

En caso de no requerir la modificación del archive general puede ser especificado como tercer parámetro la dirección donde se desee almacenar el resultado, tal y como se muestra a continuación.

```
exec("cmd.sh del 'regex' /home/me/register.log /home/me/newRegister.log ");
```

- **replace:** A diferencia del anterior donde se elimina la información especificada, se remplaza por otro contenido, por tanto se le incorpora un nuevo parámetro que especifica el texto de remplazo.

```
exec("cmd.sh replace 'regex' 'xeger' /home/me/register.log");
```

Al igual que la opción del en caso de no requerir sobrescribir el archive original se le puede especificar en este caso un cuarto parámetro que determine la dirección donde el mismo será generado.

Pese a que esta variante resuelve la problemática planteada al inicio, se recomienda realizar un estudio mas profundo en el tema que garantice mayores niveles de seguridad, así como soporte para otros entornos como *Windows*, *MacOS*, entre otros.

3.2 Servidor de Bases de Datos

En el caso de GeneSIG se definió como SGBD a utilizar el PostgreSQL, teniendo en cuenta que es un proyecto de software libre distribuido bajo licencia BSD¹³, de gran estabilidad y creado con el aporte de varios colaboradores y auspiciantes a nivel mundial bajo los estándares de ANSI-SQL 92/99. Para el soporte de datos espaciales se emplea PostGIS, el cual se integra en forma de modulo. Un aspecto que debemos de tener en cuenta es que PostGIS ha sido certificado en 2006 por el OGC¹⁴ lo que garantiza la interoperabilidad con otros sistemas también interoperables. PostGIS almacena la información geográfica en una columna del tipo *geometry*, que es diferente del homónimo "geometry" utilizado por PostgreSQL, donde se pueden almacenar la geometría en formato WKB¹⁵, aunque hasta la versión 1.0 se utilizaba la forma WKT¹⁶.

En función de iniciar el proceso de instalación se puede utilizar a través de la interfaz de

¹³ **BSD** acrónimo de Berkeley Software Distribution

¹⁴ **OGC** acrónimo de Open Geospatial Consortium

¹⁵ **WKB** acrónimo de Well-Known Binary

¹⁶ **WKT** acrónimo de Well-Known Text

Líneas de comando empleando la aplicación *apt-get* la cual se encuentra disponible en una amplia gama de distribuciones de GNU/Linux, o bien se puede emplear de forma gráfica y mas intuitiva un sistema gestor de paquetes como el Synaptic.

```
$ sudo apt-get install postgresql postgresql-client postgresql-contrib postgresql-9.1-postgis postgis pgadmin3
```

E	Paquete	Versión instalada	Última versión	Descripción
✓	postgis	1.5.3-1	1.5.3-1	Geographic objects support for PostgreSQL
✗	postgresql	9.1+122ubuntu1	9.1+122ubuntu1	object-relational SQL database (supported version)
✗	postgresql-9.1	9.1.1-1	9.1.4-0ubuntu11.10	object-relational SQL database, version 9.1 supported by the PostgreSQL community
✓	postgresql-9.1-postgis	1.5.3-1	1.5.3-1	Geographic objects support for PostgreSQL
✗	postgresql-autodoc		1.40-3	Utility to create a PostgreSQL database schema documentation
✗	postgresql-client		9.1+122ubuntu1	front-end programs for PostgreSQL (supported version)
✗	postgresql-client-8.4		8.4.8-2	front-end programs for PostgreSQL 8.4
✗	postgresql-client-9.1	9.1.1-1	9.1.4-0ubuntu11.10	front-end programs for PostgreSQL 9.1
✗	postgresql-client-common	122ubuntu1	122ubuntu1	manager for multiple PostgreSQL client versions
✗	postgresql-common	122ubuntu1	122ubuntu1	PostgreSQL database-cluster manager

Ilustración 10 Listado de paquetes relacionados con ‘postgres’ desde el *Synaptic*

Un elemento importante a destacar es que pese a que en los repositorios actuales se encuentran versiones superiores de postgres se recomienda instalar la 8.3.

3.2.1 Configuración

PostgreSQL se encuentra provisto por un gran cumulo de elementos de configuración que nos permiten adaptar este sistema gestor de bases de datos a las características de nuestros entornos necesidades, por tanto en directorio */etc/postgresql/9.1/main/* se encuentra la gran mayoría de los archivos de configuración tales como: environment, pg_hba.conf, pg_ident.conf, start.conf, pg_ctl.conf, _hba.conf.save, postgresql.conf. De estos los mas utilizados son: pg_hba.conf y postgresql.conf. Notese que la dirección de estos archivos varian en dependencia de la versión del producto.

3.2.1.1 pg_hba.conf

Este fichero se utiliza para definir como, donde y desde que sitio un usuario puede utilizar nuestro cluster PostgreSQL. Todas las lineas que empiezen con el carácter # se interpretan como comentarios. El resto debe de tener el siguiente formato:

```
[Tipo de conexión] [database] [usuario] [IP] [Netmask] [Tipo de autenticación] [opciones]
```

Dependiendo del tipo de conexión y del tipo de autenticación, ip, netmask y opciones pueden ser opcionales. Vamos a explicar un poco como definir las reglas de acceso. El tipo de conexión puede tener los siguientes valores, local, host, hostssl y hostnossll. El tipo de método puede tener los siguientes valores, trust, reject, md5, crypt, password, krb5, ident, pam o ldap

Una serie de ejemplos nos ayudaran a comprender mejor como podemos configurar diferentes accesos al cluster PostgreSQL.

Ejemplo 1 .- Acceso por tcp/ip (red) a la base de datos test001, como usuario test desde el ordenador con IP 10.0.0.100, y metodo de autenticacion md5:

```
host test001 test 10.0.0.100 255.255.255.255 md5
```

Esta misma entrada se podria escribir tambien con la mascara de red en notacion CIDR:

```
host test001 test 10.0.0.100/32 md5
```

Ejemplo 2 .- Acceso por tcp/ip (red) a la base de datos test001, como usuario test desde todos los ordenadores de la red 10.0.0.0, con mascara de red 255.255.255.0 (254 ordenadores en total) y metodo de autentificacion md5:

```
host test001 test 10.0.0.0 255.255.255.0 md5
```

Esta misma entrada se podria escribir tambien con la mascara de red en notacion CIDR:

```
host test001 test 10.0.0.0/24 md5
```

Ejemplo 3 .- Acceso por tcp/ip (red), encriptado, a todas las bases de datos de nuestro cluster, como usuario test desde el ordenador con IP 10.0.0.100, y el ordenador 10.1.1.100 y metodo de autentificacion md5 (necesitamos dos entradas en nuestro fichero pg_hba.conf):

```
hostssl all test 10.0.0.100 255.255.255.255 md5  
hostssl all test 10.1.1.100 255.255.255.255 md5
```

Ejemplo 4.- Denegar el acceso a todos las bases de datos de nuestro cluster al usuario test, desde todos los ordenadores de la red 10.0.0.0/24 y dar acceso al resto del mundo con el metodo md5:

```
host all test 10.0.0.0/24 reject  
host all all 0.0.0.0/0 md5
```

Asi podriamos seguir jugando con todas las posibilidades que nos brinda este fichero de configuracion. Por supuesto que las bases de datos y usuarios usados en este fichero tienen que existir en nuestro cluster para que todo funcione y algunos de los parametros solo se pueden usar si hemos compilado con las opciones pertinentes en el proceso de instalacion (por ejemplo, hostssl, pam, krb5)

Para poder en produccion los cambios en este fichero tendremos que decirle a PostgreSQL que vuelva a leerlo. Basta con un simple 'reload' (/usr/local/bin/pg_ctl -D /var/pgsql/data reload) desde la linea de comandos o con la funcion pg_reload_conf() como usuario postgres desde psql, el cliente PostgreSQL.

```
[postgres@servidor]# /usr/local/bin/psql  
Welcome to psql 8.2.4, the PostgreSQL interactive terminal.  
Type: \copyright for distribution terms  
      \h for help with SQL commands  
      \? for help with psql commands  
      \g or terminate with semicolon to execute query  
      \q to quit
```

```
postgres=# SELECT pg_reload_conf();  
pg_reload_conf  
-----  
t
```

```
(1 row)
postgres=#
```

3.2.1.2 postgresql.conf

Los cambios que realicemos en este fichero afectaran a todas las bases de datos que tengamos definidas en nuestro cluster PostgreSQL. La mayoria de los cambios se pueden poner en produccion con un simple 'reload' (/usr/local/bin/pg_ctl -D /var/pgsql/data reload), otros cambios necesitan que arranquemos de nuevo nuestro cluster (/usr/local/bin/pg_ctl -D /var/pgsql/data restart).

Mas informacion sobre todos los parametros que podemos cambiar en este fichero, que afectan y como se pueden poner en produccion se puede encontrar en la seccion 17. Server Configuration de la documentacion oficial de PostgreSQL.

A continuacion vamos a ver los parametros mas importantes que deberiamos cambiar si empezamos a usar PostgreSQL para un uso serio y si queremos sacarle el maximo partido a nuestra maquina. Existen muchos mas parametros que se pueden y con el tiempo se deberan de ajustar, aqui nos vamos a centrar en los mas importantes y los cuales deberiamos cambiar antes de empezar a utilizar PostgreSQL de una manera seria.

max_connections: Numero maximo de clientes conectados a la vez a nuestras bases de datos. Deberiamos de incrementar este valor en proporcion al numero de clientes concurrentes en nuestro cluster PostgreSQL. Un buen valor para empezar es el 100:

```
max_connections = 100
```

shared_buffers: Este parametro es importantisimo y define el tamaño del buffer de memoria utilizado por PostgreSQL. No por aumentar este valor mucho tendremos mejor respuesta. En un servidor dedicado podemos empezar con un 25% del total de nuestra memoria. Nunca mas de 1/3 (33%) del total. Por ejemplo, en un servidor con 4Gbytes de memoria, podemos usar 1024MB como valor inicial.

```
shared_buffers = 1024MB
```

work_mem: Usada en operaciones que contengan ORDER BY, DISTINCT, joins, En un servidor dedicado podemos usar un 2-4% del total de nuestra memoria si tenemos solamente unas pocas sesiones (clientes) grandes. Como valor inicial podemos usar 8 Mbytes.

```
work_mem = 8MB
```

maintenance_work_mem: Usada en operaciones del tipo VACUUM, ANALYZE, CREATE INDEX, ALTER TABLE, ADD FOREIGN KEY. Su valor dependera mucho del tamaño de nuestras bases de datos. Por ejemplo, en un servidor con 4Gbytes de memoria, podemos usar 256MB como valor inicial.

```
maintenance_work_mem = 256MB
```

effective_cache_size: Parametro usado por el 'query planner' de nuestro motor de bases de datos para optimizar la lectura de datos. En un servidor dedicado podemos empezar con un 50% del total de nuestra memoria. Como maximo unos 2/3 (66%) del total. Por ejemplo, en un servidor con 4Gbytes de memoria, podemos usar 2048MB como valor

inicial.

```
effective_cache_size = 2048MB
```

checkpoint_segments: Este parametro es muy importante en bases de datos con numerosas operaciones de escritura (insert,update,delete). Para empezar podemos empezar con un valor de 64. En grandes databases con muchos Gbytes de datos escritos podemos aumentar este valor hasta 128-256.

```
checkpoint_segments = 64
```

Es muy importante tener en cuenta que al aumentar los valores por defecto de muchos de estos parametros, tendremos que aumentar los valores por defecto de algunos parametros del kernel de nuestro sistema. Informacion detallada de como hacer esto se encuentra en la seccion 16.4. Managing Kernel Resources de la documentacion oficial de PostgreSQL.

En fin, esto es solo un aperitivo de lo que podemos hacer. Con la practica y la experiencia podremos y tendremos que ajustar otros muchos parametros. Pero esto sera materia de un proximo articulo.

3.2.2 Creación de bases de datos espaciales

```
pg_dump -E SQL_ASCII -F c -f /var/www/datosbash.backup -v -h 10.12.170.130 -U ing DATOSSIG  
pg_dump -E UTF8 -F c -f /var/www/libergis.backup -v -h 10.12.170.130 -U segsig Libergis  
pg_restore --host localhost --port 5432 --username postgres --dbname template_gis --verbose "/var/www/template_gis.backup"
```

3.3 Servidor de Mapas

MapServer es un servidor de mapas, que fue concebido en un ambiente de desarrollo OpenSource para permitir construir aplicaciones que trabajen con informació geoespacial, y que además tenga la cualidad de ser accesible a través de Internet. El software trabaja incorporando otras aplicaciones OpenSource las cuales en su conjunto ofrecen una solución tecnológica de alto nivel para el tratamiento de este tipo de información, tales como: Shapelib, FreeType, Proj4, libTIFF y otros.

Una característica muy importante es que MapServer puede funcionar sobre la mayor parte de versiones de Sistemas operativos, UNIX/LINUX, Microsoft Windows XP/NT/98/95 y hasta MacOS. Además soporta variados formatos y posee importantes características de los cuales se mencionaran algunas:

- Formato vectorial: shapefiles de ESRI, PostGIS, ArcSDE de ESRI. y muchos otros vía OGR 9 .
- Formato Raster: TIFF/GeoTIFF, EPPL7 GDAL y muchos otros vía GDAL.
- Quadtree indexación espacial para shapefiles.
- Totalmente adaptable, salida con plantillas HTML
- Selección de características espaciales, como puntos, áreas, etc.
- Soporta fuentes TrueType
- Configuración vía URLs 12
- Distintas proyecciones, etc.

También se distribuye su instalación desde el sitio oficial del mismo en forma de paquete compacto denominado FGS. Este incluye las dependencias que se muestran a continuación:

- mapserver-php:
5.4.0
- mapserver-base:
5.4.0
- libstdc++-lib: 6.0.8
- libgcc-lib:1
- apache-base: 2.2.11
- expat-base: 2.0.1
- gd-lib: 2.0.35
- jpeg-lib: 6b
- freetype-lib: 2.3.9
- libpng-lib: 1.2.35
- zlib-lib: 1.2.3
- postgis-lib:1.3.5
- curl-lib: 7.19.4
- openssl-lib: 0.9.8k
- proj-lib: 4.6.1
- postgresql-lib: 8.3.7
- gdal-base: 1.6.0
- tiff-lib: 3.8.2
- libgeotiff-lib: 1.2.5
- xerces_c-base:
3.0.1
- unixODBC-base:
2.2.12
- libungif-base: 4.1.4
- libiconv-base: 1.12
- proj4_epsg42xxx-support: 4.6.1
- geos-lib :3.1.0
- libxml2-base: 2.7.3
- agg-lib: 2.4
- php-base: 5.2.9
- python_mapscript-module: 5.4.0
- python-base: 2.6.2
- gdal_ecw-module:
1.6.0
- libecw-base:
3.3.20060906

3.4 Complementarios

En este acápite se describe el proceso de instalación de algunas herramientas complementarias que son utilizadas en la plataforma. Las cuales se encuentran condicionadas principalmente por los entornos de despliegue y negocio.

3.4.1 PGrooting

pgrooting:

```
libgeos
gdal
listboost
listboost-graph
libcgal
gaul
```

4. Mapserver

Los mapas digitales aseguran la relación entre conveniencia y eficacia de las imágenes gráficas, porque estos son de tipo dinámico permitiendo que información puede ser actualizada en tiempo real. Por otra parte el arduo trabajo cartográfico siempre ha sido la colección y mantenimiento de la información subyacente. El desarrollo de los mapas digitales surgió por las necesidades de la industria. Con el desarrollo de Internet y la conmutación de *hardware*, han contribuido al uso de los mapas digitales en actividades civiles como el GPS para los sistemas de la navegación automovilística, y algunos sitios de Internet que proporcionan mapas de países, ciudades, calles entre otras aplicaciones de la cartografía digital.

Sin embargo este procesamiento informático es realmente tedioso si no se emplea un software profesional que se responsabilice de la automatización del mismo, los cuales se

denominan servidores de mapas digitales y entre los que se encuentran *Mapserver*, *Goserver*, entre otros.

Como bien se había mencionado en acápitulos anteriores, MapServer es un entorno de desarrollo en código abierto para la creación de aplicaciones SIG, con el fin de visualizar, consultar y analizar información geográfica a través de la red mediante la tecnología IMS. Su funcionamiento básico está configurado en un fichero de texto, que tiene la extensión ".map". En este fichero, los datos del mapa se organizan en capas, a su vez dividida en una o más clases, donde en cada una de las cuales se pueden definir diferentes estilos visuales. Esta estructura permite la generación de mapas con una definición de estilos muy flexible, que también puede depender de la escala del mapa.

Una aplicación SIG basada en el empleo de MapServer requiere como mínimo los siguientes recursos:

- El software MapServer.
- Un servidor HTTP (*Apache/Internet Information Server*).
- Un archivo Mapfile.
- Un archivo Template.
- Una fuente de datos SIG o MTD.

El formato salida de MapServer, dependiendo de la solicitud, puede ser gráfico (mapa, leyenda, escala, métricas, visión general) o alfanumérico (el resultado de una consulta de datos alfanuméricos o espacial). El archivo ".map" también incluye la posibilidad de fusionar la producción de una plantilla de HTML MapServer, para generar una página web de lectura fácil y agradable.

Sus características principales son:

- Se ejecuta bajo plataformas Linux/Apache y Windows (MS4W)
- Formatos vectoriales soportados: ESRI shapefiles, PostGIS, ESRI ArcSDE, GML y otros muchos vía OGR.
- Formatos raster soportados: JPG, PNG, GIF, TIFF/GeoTIFF, EPPL7 y otros vía GDAL.
- Fuentes TrueType
- Configuración "al vuelo" vía parámetros GET pasados por URL
- MapScript proporciona una API para poder acceder a las funcionalidades de MapServer mediante lenguajes de programación como PHP, Java, Perl, Python, Ruby o C#.
- Soporte de estándares interoperables y conformes con Open Geospatial Consortium, como WMS, SLD, WFS, WCS y SOS.

Un aspecto técnico que destaca a MapServer por sobre otras herramientas es la capacidad de soportar MapScript, que posibilita el acceso a la API 8 de MapServer, para ello se debe contar con lenguajes de programación tales como PHP, JAVA, PYTHON, etc. Que son capaces de trabajar con MapScript y MapServer. Gracias a esa funcionalidad es posible generar aplicaciones cubriendo necesidades específicas, es decir se puede crear una aplicación para una determinada tarea, por ejemplo, un estudio de la zonificación de los niveles de delincuencia a nivel regional por citar una idea. Además permite la conexión a diferentes motores de Bases de Datos espaciales, tanto comerciales como libres.

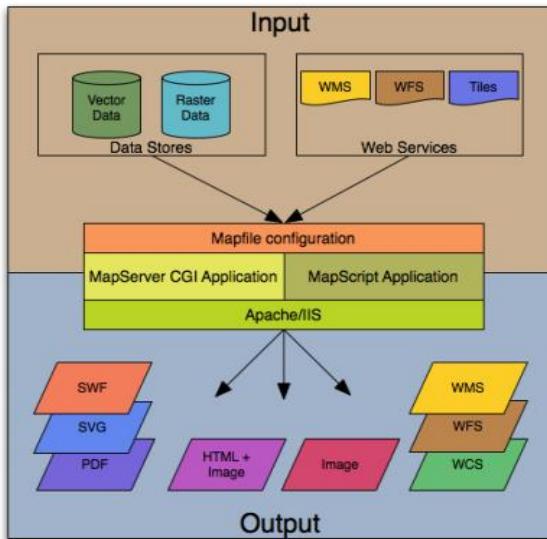


Ilustración 11 Anatomía de Mapserver

Es importante clarificar que MapServer no pretende ser un SIG en Internet, ni su objetivo es ese, sin embargo, MapServer proporciona la funcionalidad y el soporte para apoyar una variedad amplia de aplicaciones generadas por sistemas de información geográficas, con la característica principal que esta sea desplegada por medio de la Web, además de contar hoy en día con muchas de las funcionalidades que ofrecen los SIG de escritorios.

4.1 Mapfile

Es un archivo de configuración en el que se definen los recursos que serán utilizados en el servidor de mapas. Contiene información acerca de cómo se debe dibujar el mapa principal y el de referencia, la leyenda, la escala, así como el tratamiento de las consultas. Por tanto contiene todos los parámetros de entrada requeridos por el Mapserver en función de construir a través de una imagen, la representación del terreno en un plano bidimensional. En otros términos, este archivo puede ser considerado como un elemento de configuración de extensión *.map*.

Originalmente desarrollado a mediados del año 1990 en la Universidad de Minnesota, sirviendo como complemento del MapServer por lo que es liberado bajo una licencia estilo *MIT*, y puede ser utilizado en todas las principales plataformas *Windows*, *GNU/Linux*, *MacOS X*. El archivo *.map* consta de varias secciones, cada sección se inicia con el nombre de la sección y termina con la etiqueta *END*. El contenido de las secciones consiste en la definición de determinados parámetros del tipo “atributo valor”.

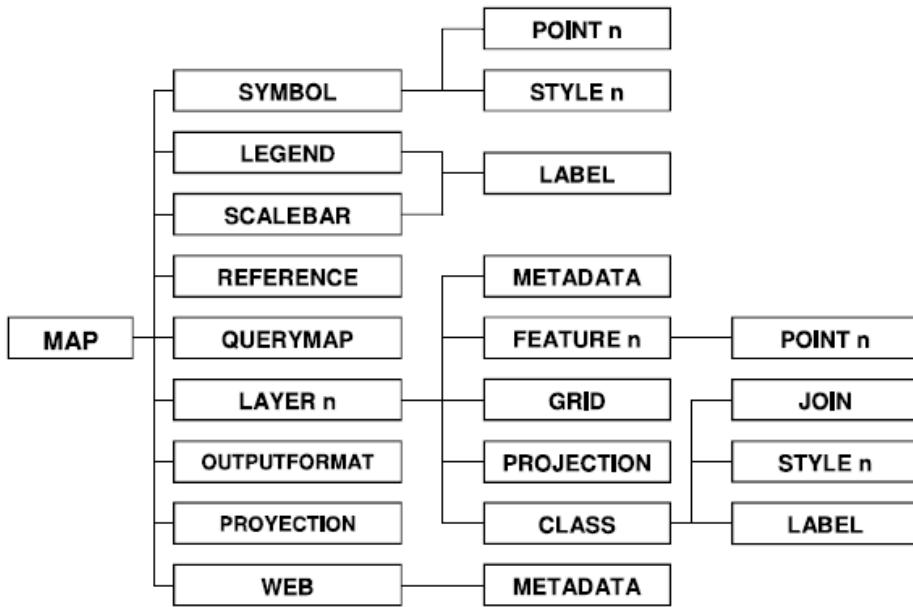


Ilustración 12 Relación entre los objetos del mapfile

El *Map* es el objeto principal y determina las propiedades del mapa en general. Contiene los objetos: *layer*, *legend*, *scalebar*, *reference*, *web*, *projection*, *metadata*. Entre las propiedades que lo componen se encuentran:

- **NAME:** Define una etiqueta nombre como identificadora para determinado Objeto, esta propiedad es utilizada internamente por MAP, LAYER, CLASS, STYLE, etc.
- **SIZE:** Determina las proporciones de la imagen resultante, si se utilizan fuentes truetype, el valor se especifica en pixels, si es bitmap se especifica en: small | large.
- **STATUS:** Establece si el mapa o la capa estará activo o no. El valor DEFAULT es siempre en ON, el valor ON permite que el objeto quede visible y el valor OFF todo lo contrario, los valores admitidos son: [ON / OFF/ DEFAULT]
- **EXTENT:** Define la extensión espacial del mapa a crear, en el sistema de referencia especificado en la sección PROJECTION, en otras palabras, especifica las dimensiones de visualización para la salida del mapa, definiendo para ello dos puntos georeferenciados, el formato de los mismos seria: [Xmin] [Ymin] [Xmax] [Ymax].
- **UNITS:** Define las unidades de las coordenadas del mapa, usado para el cómputo de la escala gráfica y escala numérica. Debe estar definido en el sistema de referencia especificado en la sección PROJECTION, estos pueden ser: [feet | ches | kilometers | meters | miles | dd]
- **IMAGECOLOR:** Esta propiedad define el color de background del mapa, los valores son RGB [Red] [Green] [Black].
- **IMAGETYPE:** Esta propiedad define el formato de salida para las imágenes, los valores pueden ser muchos, entre ellos se puede utilizar el color indexado *PNG*

similar al *GIF*, se podría usar *GIF* si se compila la librería GD con soporte para *GIF*, *WBMP* o *JPEG*, también se puede especificar otros formatos de salida *PDF*, *SWF* y *Geotiff* asumiendo que se ha compilado el soporte para los mismos y que el *OUTPUTFORMAT* es de este tipo, en resumen pueden ser: [*gif* | *png* | *jpeg* | *wbmp* | *gtiff* | *swf* | *userdefined*]

- **SHAPEPATH:** Se le especifica el nombre del directorio donde se almacenan los datos geográficos. Esta es la ruta para los datos de que serán visualizados en los diferentes layers.
- **FONSET:** Es un parámetro del objeto MAP que determina la ruta completa del archivo truetype fontlist. Este archivo lista cada una de las fuentes disponibles.
- **OFFSITE:** Este parámetro indica a MapServer los valores de pixel para renderizar como background.

El objeto LAYER determina las propiedades para una fuente de datos, se pueden crear tantos objetos *layer* como sean necesarios aunque el límite es de 100 por defecto. Contiene objetos como: *projection*, *metadata*, *class*. Entre las propiedades que lo componen se encuentran:

- **NAME:** Define una etiqueta nombre como identificadora para determinado Objeto, esta propiedad es utilizada internamente por *map*, *layer*, *class*, *style*, etc.
- **GROUP:** Se le especifica un Nombre de un grupo o conjunto de capas.
- **TYPE:** Determina la forma de representación de las capas o el dibujado de la información, el valor *raster* se refiere a las imágenes mapas de bit, los valor posibles son: [*point* | *line* | *polygon* | *circle* | *annotation* | *raster* | *query*]
- **TYPERASTER:** Cuando se usan datos raster se deberá utilizar el valor *raster* del parámetro *type*, como opuesto al *polygon*, *line* o *point* que son valores de datos vectoriales y *annotation* para la identificación de etiquetas, estos pueden ser: [*gif* | *png* | *jpeg* | *wbmp* | *gtiff* | *swf* | *userdefined*]
- **DATA:** Determina la fuente de datos a utilizar, en caso de que esta se encuentre en formato Shapefiles se especifica el nombre del fichero y no requiere del atributo *connection*, en caso de que se utilice el formato PostGIS, entonces se especifica el campo geoespacial de la base de datos a utilizar y los atributos *connection* y *connectiontype*.
- **CONNECTION:** Para cada capa de información que contendrá el servicio, deberá definirse el parámetro *connection*, especificando de esta forma la conexión a la fuente de datos.
- **CONNECTIONTYPE:** Define el tipo de conexión, que por defecto es local, este parámetro debe incorporarse en el caso que se desee incluirse una capa remota, los posibles valores son: [*local* | *sde* | *ogr* | *postgis* | *oraclespatial* | *wms*]
- **CLASSITEM:** Se le especifica el nombre del item en tabla de atributos a usar como filtro para aplicar el objeto *class* con la finalidad de especificar que atributos se utilizan para separar un objeto de la clase.
- **LABELITEM:** Se le especifica el nombre del item en tabla de atributos a usar como anotación.
- **HEADER:** Se le especifica el nombre del archivo Plantilla para ser usado como encabezado de la plantilla de respuesta a consultas. (modo query).
- **FOOTER:** Se le especifica el nombre del archivo Plantilla para ser usado como cierre de la plantilla de respuesta a consultas. (modo query)

- **TRANSPARENCY:** Establece un nivel de transparencia para la capa. El valor es un porcentaje de 0 a 100 donde 100 es opaco y 0 es totalmente transparente.
- **TOLERANCE:** Sensibilidad para las consultas basadas en puntos.
- **TILEINDEX:** Archivo Shapefile que contiene los rectángulos envolventes de cada una de las piezas que forman el mosaico.
- **PROCESSING:** Este parámetro del objeto *layer* se le incorpora a la versión 4,0 de MapServer, puede tener muchos valores, ejemplo: “bands=1,2,3”, en este caso se está usando la selección de una banda multiespectral en la imagen a mostrar.

El objeto CLASS determina un conjunto de propiedades específicas para un objeto *layer*. Contiene objetos como: *style*. Entre las propiedades que lo componen se encuentra:

- NAME: Nombre a ser utilizado en la generación de leyenda para esta clase. Si no se incluye ningún nombre, no aparecerá esta clase en la leyenda.
- COLOR: En el objeto LABEL, COLOR especifica el control de etiqueta de texto.
- OUTLINECOLOR: Color para la línea externa de polígonos. No es soportado por líneas. Este valor está dado en RGB.
- EXPRESSION: Esta propiedad es utilizada para filtrar capas para registros específicos en el conjunto de datos y de esta forma especificar a cuáles elementos se le aplica esta clase, existen tres tipos diferentes de expresiones:
 - **Comparaciones de cadena:** Como el nombre sugiere se comprueba que los valores de los atributos sean iguales a algún valor, por tanto un solo atributo se compara con un valor de cadena, las comparaciones de cadena son la forma más sencilla de las expresiones de MapServer y la opción más rápida.
 - **Expresiones regulares:** El trabajo con expresiones regulares en MapServer es de manera similar a la comparación de cadenas, pero permite operaciones más complejas, son más lentas que las puras comparaciones de cadena, pero podría ser aún más rápidas que las expresiones lógicas. Al igual que las comparación de cadenas usa expresiones regulares, *filteritem* o *classitem* tiene que definirse, respectivamente, por tanto un único atributo es acompañada de una expresión regular.

Una expresión regular consiste en una serie de caracteres con un significado especial y caracteres que se interpretan como son.

- Caracteres alfanuméricos: [AZ / az / 09]
- Caracteres con significados especiales:
 - (“.”) Coincidirá con un solo carácter.
 - [] Se utiliza para agrupar.
 - {,},* Se utiliza para especificar con qué frecuencia algo debe coincidir.
 - ^ Se utiliza para marcar el comienzo.
 - \$ Se utiliza para marcar el final del valor.
 - \ Se utiliza para quitar el significado especial.
- **Logicas “Expresiones de MapServer”:** Sintácticamente una expresión lógica es todo encapsulado en ciclos, estas toman valores lógicos como sus valores de entrada y retorna valores lógicos, por tanto uno o más atributos se comparan utilizando este tipo de expresiones la cual puede ser

«verdadera» o «falsa».

Para utilizar una expresión de MapServer con un valor de tipo *filter* o *expression*, esta ha de ser finalmente de un valor lógico.

- **Conjunción:** Se convertirá en verdadero cuando las dos expresiones lógicas en el interior son verdaderas.
 - ((...) AND (...))
 - ((...) && (...))
- **Disyunción:** Se convertirá en verdadero cuando al menos una de las dos expresiones lógicas en el interior sea verdadera.
 - ((...) OR (...))
 - ((...) || (...))
- **Negación:** Se convertirá en verdadero cuando la expresión sea falsa.
 - NOT (...)
 - ! (...)
- **Igualdad entre cadenas:** Se convertirá en verdadero cuando ambas cadenas son iguales. Esta operación es idéntica a la cadena de comparación de MapServer descrita anteriormente.
 - ("String1" eq "String2")
 - ("String1" == "String2")
 - ("String1" = "String2")
- **Desigualdad entre cadenas:** se convertirá en verdadero cuando ambas cadenas no son iguales.
 - ("String1" != "String2")
 - ("String1" ne "String2")
- **Similitud TextoToquen:** Será cierto cuando "texto1" es igual a uno de los tokens dados.
 - ("String1" IN "token1,token2,...,tokenN")
- **Similitud TextoExpresión:** Se convertirá en verdad, cuando "String1" coincida con la expresión regular "regexp". Esta operación es idéntica a la expresión regular coincidente que se describe anteriormente.
 - ("String1" =~ /regexp/)

La etiqueta STYLE determina un conjunto de propiedades específicas para un objeto de tipo CLASS. Entre las propiedades contenidas dentro de la etiqueta se encuentra: *label*, *angle*, *backgroundcolor*, *backgroundshadowcolor*, *backgroundshadowsize*, *color*, *font*, *force*, *maxsize*, *minsize*, *mindistance*, *offset*, *outlinecolor*, *partial*, *position*, *shadowcolor*, *shadowsize*, *size*, *type*, *imagecolor*, *keysize*, *keyspacing*, *status*, *symbolset*, *buffer*, *mindistance*, *partials*.

El LABEL puede ser empleado por la mayoría de los restantes objetos, pues este se encarga de describir las características asociados al campo de texto que se muestra en forma de tituto para determinado recurso sobre el mapa. Entre las propiedades que lo describen se encuentran:

- **ANGLE:** Ángulo en grados, para dibujar la etiqueta o *auto* para que el software coloque la etiqueta alineada a la línea (aplicable solo a capas lineales).
- **BACKGROUNDCOLOR:** Color con el que se dibujará el rectángulo de fondo. Por defecto no se coloca rectángulo [R][G][B]

- **BACKGROUNDSHADOWCOLOR:** Color de la sombra del rectángulo. Por defecto no se coloca.[R] [G] [B]
- **BACKGROUNDSHADOWSIZE:** Cuan lejos del rectángulo se dibujará la sombra, [x] [y]
- **COLOR:** Color del texto, [R] [G] [B]
- **FONT:** Nombre del tipo de letra como fue definido en FONTSET.
- **FORCE:** Evita que las etiquetas se superpongan. true | false
- **MAXSIZE:** Tamaño máximo de la fuente.
- **MINSIZE:** Tamaño mínimo de la fuente.
- **MINDISTANCE:** Mínima distancia entre etiquetas.
- **OFFSET:** Separación de la etiqueta del punto etiquetado. [x] [y]
- **OUTLINECOLOR:** Color de la línea exterior de un píxel del texto. [R] [G] [B]
- **PARTIAL:** Pueden las etiquetas continuar fuera del mapa?. los valores aceptados son: [true | false]
- **POSITION:** Posición que ocupará la etiqueta respecto del punto etiquetado, los valores aceptados son: [ul | uc | ur | cl | cc | cr | ll | lc | lr | auto]
 - **ul:** superior izquierda
 - **uc:** superior centro
 - **ur:** superior derecha
 - **cc:** centro
 - **cr:** centro derecha
 - **cl:** centro derecha
 - **ll:** inferior izquierda
 - **lc:** inferior centro
 - **lr:** inferior derecha
- **SHADOWCOLOR:** Color de la sombra. [R] [G] [B]
- **SHADOWSIZE:** Separación de la sombra en píxeles. [x] [y]
- **SIZE:** Tamaño del texto. los valores aceptados son: [integer] | [tiny | small | medium | large | giant]
- **TYPE:** Tipo de la fuente a usar. los valores aceptados son: [bitmap|truetype]

Para definir la proyección de los mapas es necesario especificar dos objetos *projection*, uno en el objeto *map* para la generación de la imagen de salida y otro para cada capa en el objeto *layer*. Cada capa puede tener un sistema de referencia diferente y el servidor de mapas se encargará de proyectarla al sistema especificado, teniendo en cuenta que MapServer utiliza la librería PROJ4 “Geographic Projection Library” para tal fin, el sistema de referencia y proyección pueden ser definido de dos maneras, una es especificando los parámetros de la proyección y otra utilizando la codificación del European Petroleum Survey.

Ejemplo de definición de UTM zona 15, NAD83:

```
PROJECTION
  "proj=utm"
  "ellps=GRS80"
  "zone=15"
  "north"
  "no_defs"
END
```

Para definir coordenadas Geográficas: PROJECTION "proj=latlong" END

Utilizando la codificación del European Petroleum Survey Group (EPSGP): PROJECTION

"init=epsg:23030" END

El SCALEBAR define cómo se construirá la escala gráfica. Comienza con la palabra *scalebar* y termina con *end*. Propiedades contenidas dentro de la etiqueta:

- **STYLE:** Puede elegirse entre dos estilos [o | 1] de escala gráfica.
- **STATUS:** Define el estado del *scalebar*, el valor por defecto es off, los posibles valores son: [on | off | embed]
 - **on:** la escala gráfica será generada
 - **off:** la escala gráfica no será generada
- **embed:** la escala gráfica se generará embebida en el mapa generado.
- **SIZE:** Tamaño en píxeles de la escala gráfica. El etiquetado (labeling) no está considerado dentro de estos valores. sintaxis: [x][y]
- **COLOR:** Color en que se dibujará la escala gráfica. [R] [G] [B]
- **UNITS:** Unidades de la escala gráfica. Grados decimales (dd) no es una unidad válida. El valor por defecto es miles. [feet | inches | kilometers | meters | miles]
- **INTERVALS:** Número de intervalos en que se dividirá la escala gráfica. Por defecto es 4.
- **TRANSPARENT:** Permite que el fondo de la escala gráfica sea transparente. Por defecto es off, [on | off]
- **POSITION:** Posición que ocupará la escala gráfica embebida, por defecto es Ir, los valores permitidos son: [ul | uc | ur | ll | lc | lr]
 - **ul:** superior izquierda
 - **uc:** superior centro
 - **ur:** superior derecha
 - **ll:** inferior izquierda
 - **lc:** inferior centro
 - **lr:** inferior derecha
- **BACKGROUNDCOLOR:** Color usado para el fondo de la escala gráfica, no para el fondo de la imagen. [R] [G] [B]
- **IMAGECOLOR:** Color con el que se inicializará la escala gráfica. [R] [G] [B]
- **OUTLINECOLOR:** Color de la línea exterior de cada intervalo. Para que los intervalos no presenten línea exterior debe colocarse -1 -1 -1. [R] [G] [B]

El recurso REFERENCE define cómo será creado el mapa de referencia. Este es un mapa que comprende la extensión total de la zona que incluirá el servicio de WMS, sobre él se representará una marca en la zona que se visualiza actualmente, actualizándose interactivamente. También es posible realizar un *clic* sobre determinado sector del mapa de referencia y Mapserver generará el mapa de dicha zona. En las consultas puede generar se un mapa de referencia, resaltándose en el mismo el punto(x,y) , la zona geográfica o la entidad consultada. Propiedades contenidas dentro de la etiqueta:

- **IMAGE:** Se le especifica el nombre completo del archivo de la imagen que será usada para generar el mapa de referencia.
- **EXTENT:** Extensión espacial de la imagen de referencia, en el sistema de referencia definido en la sección PROJECTION, sintaxis: [Xmin] [ymin] [xmax] [ymax]
- **SIZE:** Tamaño en píxeles de la imagen de referencia. sintaxis: [x] [y]
- **STATUS:** Define el estado del *reference*, el valor por defecto es off, valores permitidos [on | off]
 - **on:** el mapa de referencia será generado
 - **off:** el mapa de referencia no será generado

- **MARKER:** Definición de un símbolo a utilizar cuando el recuadro sea demasiado pequeño de acuerdo a los valores asignados a *minboxsize* y *maxboxsize* valores permitidos [integer / string]
- **MARKERSIZE:** Define el tamaño del símbolo a utilizar en reemplazo del recuadro.
- **MINBOXSIZE:** Si el recuadro es más pequeño que *minboxsize* se lo reemplazará por el símbolo definido en *marker*.
- **MAXBOXSIZE:** Si el recuadro es más grande que *maxboxsize* no se dibujará ninguna marca.
- **COLOR:** Color en que se dibujará el recuadro de referencia. Para que dicho rectángulo o marca no se encuentre relleno deberá colocarse –1 –1 –1. El valor por defecto es 255 0 0 (red). [R] [G] [B]
- **OUTLINECOLOR:** Color de la línea exterior del recuadro de referencia. Para no incluir línea exterior debe colocarse –1 –1 –1. [R] [G] [B]

El objeto WEB define cómo operará la interface Web. Contiene objetos como: METADATA. Propiedades contenidas dentro de la etiqueta:

- **HEADER:** Nombre del archivo Plantilla para ser usado como encabezado de la plantilla de respuesta a consultas. (modo query)
- **TEMPLATE:** Nombre del archivo plantilla a utilizar en la que se representarán los resultados de peticiones. Página web visible por el usuario.
- **FOOTER:** Nombre del archivo Plantilla para ser usado como cierre de la plantilla de respuesta a consultas. (modo query)
- **MINSCALE:** Escala mínima para la cual la interfase es válida. Cuando un usuario peticiona un mapa a escala más pequeña, MapServer retorna el mapa a esta escala.
- **MAXSCALE:** Escala máxima para la cual la interfase es válida. Cuando un usuario peticiona un mapa a escala más grande, MapServer retorna el mapa a esta escala
- **IMAGEPATH:** Nombre del directorio donde se almacenarán los archivos e imágenes temporales. Debe terminar con “/”.
- **IMAGEURL:** URL del IMAGEPATH. Es el URL que seguirá el web browser para buscar la imagen temporal.
- **EMPTY URL** para mostrar a los usuarios cuando ante una consulta vacía o un fallo.

El METADATA deberá ser incluido tanto en el objeto MAP, como en cada LAYER. En el primer caso contendrá metadatos en general del servicio, y en el segundo caso, metadatos específicos para cada capa de información. Luego el servidor WMS/WFS se basará en estos metadatos para confeccionar el archivo de capacidades.

- **wms_srs:** Lista de espacio delimitado de proyección de códigos EPSG soportados por el servidor remoto, normalmente la recibe desde las capacidades de salida del servidor. Este valor debe ser en mayúsculas para evitar problemas en caso de que exista una plataforma sensible y se utiliza para fijar el parámetro SRS WMS URL.
- **wms_name:** Lista de capas separadas por comas que se descargan del servidor remoto.WMS. Este valor se utiliza para configurar los parámetros de las capas y de las URL de las QUERY_LAYERS WMS.
- **wms_server_version:** La versión del protocolo WMS apoyado por el servidor remoto.WMS y que será utilizado para la emisión de solicitudes GetMap.
- **wms_format:** Formato de imagen para su uso en las solicitudes GetMap. Si se proporciona wms_formatlist entonces wms_format es opcional y MapServer cogerá

el primer formato soportado en wms_formatlist para su uso en las solicitudes GetMap. Si tanto wms_format y wms_formatlist se proporcionan, wms_format tiene prioridad. Los Servidores WMS sólo soportan formatos que forman parte de las bibliotecas GD / GDAL.

El QUERYMAP define un mecanismo para asignar los resultados de una consulta. Propiedades contenidas dentro de la etiqueta:

- **COLOR:** Color de las características que están destacados. El valor por defecto es amarillo. [R] [G] [B]
- **STATUS:** Activar o no la consulta, valores permitidos [on / off]
- **SIZE:** Tamaño del mapa en píxeles. El valor por defecto es el tamaño definido en el objeto de mapa. Sintaxis: [x] [y]
- **STYLE:** Establece como serán manipuladas las características seleccionadas. Las capas no consultadas serán extraídas como de costumbre. valores permitidos: [normal / hilit / selected]

LEGEND: Para que el CGI de MapServer pueda generar la simbología automáticamente es necesario incluir dentro del archivo .map la sección LEGEND. MapServer genera la leyenda o simbología de las capas visualizadas a partir de las clases definidas (CLASS) en cada capa de información. Es una imagen, cuyo formato depende del formato definido para la creación del mapa. La sección comienza con la palabra LEGEND y finaliza con END.

- **IMAGECOLOR:** Color con el que se inicializará la leyenda. [R] [G] [B]
- **KEYSIZE:** Tamaño en píxeles de cada símbolo a crear. El valor por defecto es 20 por 10 píxeles. [x] [y]
- **KEYSPACING:** Espacio en píxeles, de separación entre cada símbolo ([y]) y entre símbolos y etiqueta ([x]). [x] [y]
- **POSITION:** Posición que ocupará la leyenda embebida, por defecto toma el valor lr, los posibles valores son: [ul | uc | ur | ll | lc | lr]
 - **ul:** superior izquierda
 - **uc:** superior centro
 - **ur:** superior derecha
 - **ll:** inferior izquierda
 - **lc:** inferior centro
 - **lr:** inferior derecha
- **STATUS:** Define el estado de la leyenda, los valor posibles son: [on | off | embed]
 - **on:** la leyenda será generada
 - **off:** la leyenda no será generada
 - **embed:** la leyenda se generará embebida en el mapa generado.

Un elemento importante a destacar y que puede ayudar mucho en función de agilizar el desarrollo, es que en caso que se requiera visualizar el contenido un mapa determinado basado en el *mapfile* que nos encontramos desarrollando, se puede emplear la herramienta *shp2img*, ejecutando un comando similar a *shp2img -m mymapfile.map -o test.png*, para mayor comprensión de este tema refiérase al acápite 9.3 *shp2img*.

4.2 Modo CGI

Como bien sea habría descrito en acápitres anteriores el Mapserver provee un recurso que

le permite servir mapas digitales vía CGI. En función de garantizar cierto grado de interactividad entre los clientes y el propio servidor, se definen un conjunto de parámetros los cuales generalmente son especificados en la propia *url*, este método de envío es conocido como *GET* y aunque también soporta otros como el *POST*, son menos utilizados. A continuación se procede a describir cada una de las opciones que pueden ser especificadas en la petición.

- **BUFFER:** Define el valor de distancia o rango de acción en el sistema de coordenadas para el archivo de mapa, se utiliza junto con *mapxy* para crear un punto nuevo mapa.
- **CONTEXT:** Permite definir la ruta de acceso a un archivo de contexto, la cual es relativa al archivo de mapa en que se utilizará, también puede ser especificada desde una URL.
- **ICON:** Es especificada en caso de definir el par clave valor *MODE=legendicon*, en función de generar un ícono para una capa determinada, la propiedad *classindex* es opcional y toma por defecto valor 0. Sintaxis *[layername], [classindex]*
- **ID:** Por defecto MapServer genera un identificador de sesión único, basado en la hora del sistema y la identificación de procesos. Este parámetro permite sobrescribir el valor predeterminado.
- **IMG:** Define el nombre asociado al componente que se encarga de representar la imagen del mapa, con el objetivo de controlar los registros de clic por el usuario, arrojando las coordenadas en pixeles para cada uno de estos.
- **IMGBOX:** Coordenadas en pixeles asociadas a los puntos extremos que componen una selección rectangular sobre la imagen del mapa. Sintaxis *[x1] [y1] [x2] [y2]*
- **IMGEXT:** Define la extensión espacial de la imagen publicada. Sintaxis *[minx] [miny] [maxx] [maxy]*
- **IMGSHAPE:** Geometría conformada por un listado de coordenadas expresadas en pixeles, el cual nos es mas que una forma de polígono arbitrario para ser usada para fines de consulta. Sintaxis *[x1 y1 x2 y2 x3 y3 ...]*
- **IMGSIZE:** Define las dimensiones expresadas en pixeles de la imagen resultante, definida por el par columnas, filas. Sintaxis *[cols] [rows]*
- **IMGXY:** Coordenadas en pixeles de un clic del ratón. Esta opción es usualmente empleada por implementaciones de Java para Mapserver. Sintaxis *[x] [y]*
- **LAYER:** Define el nombre de una capa tal como aparece en el archivo de mapa, que será activada.
- **LAYERS:** Define los nombre de las capas que serán activadas para su posterior visualización, las cuales deberán estar separadas por espacios.
- **MAP:** Define la ruta relativa al directorio CGI, del archivo de mapa que se utilizará.
- **MAPEXT:** Define la extensión territorial que será visualizada, su sintaxis corresponde a la siguiente forma *[minx] [miny] [maxx] [maxy]*. De forma alternativa se le puede especificar una estructura de tipo *shape*.
- **MAPSIZE:** define el tamaño en pixeles de la imagen que se creará. Útil para permitir a los usuarios cambiar la resolución del mapa de salida dinámica, sintaxis *[cols] [rows]*.
- **MAPSHAPE** Geometría conformada por un listado de coordenadas expresadas en pixeles, el cual nos es mas que una forma de polígono arbitrario para ser usada para fines de consulta. Sintaxis *[x1 y1 x2 y2 x3 y3 ...]*. Se utiliza con los modos NQUERY y NQUERYMAP.
- **MAPXY:** Define un punto en el mismo sistema de coordenadas que el shapefiles empleado, para ser utilizada en conjunto con un *buffer* o una escala para la

construcción de un *extent* determinado. Se puede establecer un *shape* como una opción alternativa. En este caso *mapextent* se establece en la medida de la forma seleccionada, utilizándose principalmente para las consultas, sintaxis *[x] [y]*.

- **MINX | MINY | MAXX | MAXY:** Propiedades numéricas que definen los valores máximos y mínimos de las componentes de una coordenada.
- **MODE:** Define el modo de operación. Las siguientes opciones que se describen a continuación son soportadas:
 - **BROWSE:** Interfaz completamente interactiva, este es el modo por defecto.
 - **QUERY:** Una búsqueda espacial, como obtener las geometrías más próximas, se activa mediante un clic en un mapa.
 - **NQUERY:** Una búsqueda espacial, en función de encontrar todas las geometrías, se activa mediante un clic o por el usuario definir la selección rectangular sobre el mapa.
 - **ITEMQUERY:** Una búsqueda de texto sobre los atributos del *data*, se activa mediante un *QString* para una capa determinada, retornando la primera coincidencia.
 - **ITEMNQUERY:** Una búsqueda de texto sobre los atributos del *data*, se activa mediante un *QString* para una capa determinada, retornando todas las coincidencias.
 - **FEATUREQUERY:** Una búsqueda espacial que toma como referencia una propiedad particular de una capa determinada en función de consultar el resto.
 - **FEATURENQUERY:** Una búsqueda espacial que toma como referencia multiples propiedades de una capa determinada en función de consultar el resto.
 - **ITEMFEATUREQUERY:** Búsqueda de texto sobre el atributo *data* de la capa, empleando un *qstring*, además debe ser especificado la propiedad *slayer*. Los resultados obtenidos pueden ser aplicados para activar otras búsquedas sobre las restantes capas, limitando la salida a partir del empleo del parámetro *qlayer*.
 - **ITEMFEATURENQUERY:** Similar al anterior a diferencia del anterior el cual retorna la primera coincidencia, en este caso se devuelven todas las encontradas.
 - **LEGENDICON** Permite obtener un ícono para la leyenda. El parámetro **ICON** también debe ser utilizado para especificar el nombre de la capa y un índice de clase. Por otra parte el indice de la clase es opcional y por defecto toma valor 0. A continuación se muestra un ejemplo de su uso:
<http://mapserv.cgi?map=/ms4w/apps/gmap/htdocs/gmap75.map&MODE=legendicon&ICON=popplace,0>
 - **MAP:** Retorna el mapa generado, empleándose principalmente con el uso de la etiqueta *HTML* denominada **.
 - **REFERENCE:** Retorna el mapa de referencia generado, empleándose principalmente con el uso de la etiqueta *HTML* denominada **.
 - **SCALEBAR:** Retorna la imagen asociada a la escala, empleándose principalmente con el uso de la etiqueta *HTML* denominada **.
 - **LEGEND:** Retorna la imagen asociada a la leyenda, empleándose principalmente con el uso de la etiqueta *HTML* denominada **.
 - **ZOOMIN:** Cambie al modo de exploración con *ZOOMDIR=1*
 - **ZOOMOUT:** Cambie al modo de exploración con *ZOOMDIR=-1*
 - **INDEXQUERY:** Permite obtener geometrías, a partir de los valores asociados a las propiedades: *shapeindex* y *tileindex*.
 - **SHAPEINDEX:** Esta propiedad es requerida, al contrario de *tileindex*, siendo utilizada únicamente sobre los *tiled* generados a partir de una fuente de

datos de tipo *shapefile* especificada para una *layer* determinada.

- **QLAYER:** Define el nombre de la capa que se debe consultar como aparece en el archivo de mapa. En caso de no ser especificada se tomaría todas las que estén activas.
- **QITEM:** Define el nombre de una propiedad de una capa determinada que será consultada. El parámetro es opcional y se utiliza junto con el *qstring* para consultas de atributos.
- **QSTRING:** Expresión que define los atributos de consulta.
- **QUERYFILE:** Se le especifica un nombre de archivo, utilizándose generalmente con el modo *browse* o *nquery*. Esta opción identifica un archivo de consulta para cargar ante cualquier tratamiento regular de los datos producto de una búsqueda. En el modo *nquery* garantiza el acceso a cualquiera de las plantillas que se utilizan normalmente en la presentación de la consulta, por lo que tiene acceso a los mapas de consulta e información de atributos.
- **REF:** Define el nombre asociado con la imagen del mapa de referencia utilizado para registrar los registros de clic producidos por el usuario.
- **REFXY:** Define las coordenadas en pixeles asociadas a un punto especificado por el usuario, especificándose con la siguiente sintaxis: *[x] [y]*
- **SAVEQUERY:** Cuando se utiliza cualquiera de los modos de consulta, le indica al MapServer como guardar los resultados obtenidos en un archivo temporal para su uso en operaciones posteriores (ver *queryfile*). Es útil para realizar consultas persistentes.
- **SCALEDENOM:** Define el valor de la escala en función de crear un nuevo mapa. Se utiliza con la propiedad *mapxy*, este valor se da como el denominador de la fracción de escala real, por ejemplo, para un mapa a una escala de 1:24,000 se debe especificar el valor 24000. Fue implementado en MapServer a partir de la versión 5.0, con el objetivo de sustituir el parámetro *scale*.
- **SCALE:** Esta propiedad es obsoleta desde MapServer en su versión 5.0, el parámetro correcto a utilizar es *scaledenom*.
- **SEARCHMAP:** En no pocas ocasiones se requiere realizar acciones de navegación en función de realizar determinada consulta sobre la información cartográfica, en estos casos se debe especificar, permitiendo calcular el nuevo *extent* antes de producirse la petición, es muy utilizado en el modo denominado *nquery*.
- **SHAPEINDEX:** Se emplea de conjunto con *indexquery*, en función de indexar las consultas.
- **SLAYER:** Permite seleccionar la capa sobre la cual ser realizarán búsquedas geométricas para cualquier modo de consulta especificado.
- **TILEINDEX:** Empleado para indexar consultas, principalmente para las capas basadas en *tiled shapefile*, se utiliza de conjunto con la propiedad *indexquery*
- **ZOOM:** Permite la relación del *zoom*. Los valores mayores que 0 resulta en un acercamiento, 0 constituyen un *pan*, y valores inferiores a cero son para alejar el *zoom*. Un valor de 2 significa "acercar dos veces". *zoom* se puede utilizar como un atajo para la combinación *zoomdir* / *zoomsize*
- **MINZOOM/MAXZOOM:** Define los límites de la acción denominada como *Zoom*, por defecto toma valores comprendidos entre (-25/25)
- **ZOOMDIR:** Define la dirección de la acción *zoom* y toma valores comprendidos entre [1 | 0 | -1]
- **ZOOMSIZE:** Define el valor absoluto de la magnitud del *zoom*, se utiliza de conjunto con *zoomdir*.

4.2.1 Confeccionando un mapa básico

En función de confeccionar un mapa con los elementos mínimos que permita obtener una imagen descriptiva del mismo, se crea el ejemplo que se muestra a continuación:

```
MAP
  NAME Mundo
  SIZE 600 400
  EXTENT 87.9964 19.0216 63.114 1.19528

  WEB
    IMAGEPATH "ms_tmp/img/"
    IMAGEURL "ms_tmp/url/"
  END

  LAYER
    NAME "dpa"
    STATUS ON
    DATA "GeoDatos.shp"
    TYPE POLYGON

    CLASS
      STYLE
        COLOR 200 150 2
        OUTLINECOLOR 0 0 0
      END
    END
  END
END
```

Obsérvese que un atributo indispensable lo constituye el valor de la propiedad *extent*, de lo contrario no se sabría que área geográfica contendría dicha imagen. Por otra parte es obligatorio definir al menos una capa, definiéndole un estilo que describa como será visualizado dicha información.

Para visualizar su contenido en un navegador sería tan simple como invocar url que se muestra a continuación, especificándoles como mínimo los parámetros: *mode*, *map*, *layers* <http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&LAYERS=all>.

4.2.2 Inclusión de ficheros externos

En el apartado anterior se describen los elementos esenciales que deben ser incluidos en el fichero para la caracterización de mapas en Mapserver. Sin embargo en no pocas ocasiones nos encontramos con perfiles cartográficos en extremo complejos y cargados de diversos tipos de geometrías. Siendo esto una de las cosas que hacen engoroso la confección de mapfile, así como su posterior mantenimiento. Por tanto ante dicha problemática surge como alternativa llevar a cabo un proceso de fragmentación de los elementos que componen el mismo a través de distintos recursos tales como: *fontset*, *symbolset*, *include*. A continuación se muestra un ejemplo que ilustra como poner en práctica tal solución.

```
MAP
  EXTENT -85.1920574440399 17.7331838565022 -73.8079425559601 26.2668161434978
  IMAGECOLOR 220 255 255
  IMAGETYPE png24
  SIZE 640 480
  STATUS ON
  UNITS METERS
  NAME "Cuba"

  FONTSET "fonts/geolocal.font"
```

```

SYMBOLSET "geolocal.sym"

OUTPUTFORMAT
    NAME "png24"
    MIMETYPE "image/png; mode=24bit"
    DRIVER "GD/PNG"
    EXTENSION "png"
    IMAGEMODE "RGB"
    TRANSPARENT FALSE
END

PROJECTION
    "init=epsg:4267"
END

LEGEND
    IMAGECOLOR 255 255 255
    KEYSIZE 20 10
    KEYSPACING 5 5
    LABEL
        SIZE MEDIUM
        TYPE BITMAP
        BUFFER 0
        COLOR 0 0 0
        FORCE FALSE
        MINDISTANCE -1
        MINFEATURESIZE -1
        OFFSET 0 0
        PARTIALS TRUE
    END
    POSITION LL
    STATUS OFF
END

QUERYMAP
    COLOR 255 255 0
    SIZE -1 -1
    STATUS OFF
    STYLE HILITE
END

SCALEBAR
    ALIGN CENTER
    COLOR 0 0 0
    IMAGECOLOR 255 255 255
    INTERVALS 4
    LABEL
        SIZE MEDIUM
        TYPE BITMAP
        BUFFER 0
        COLOR 0 0 0
        FORCE FALSE
        MINDISTANCE -1
        MINFEATURESIZE -1
        OFFSET 0 0
        PARTIALS TRUE
    END
    POSITION LL
    SIZE 200 3
    STATUS OFF
    STYLE 0
    UNITS MILES
END

WEB
    IMAGEPATH "log/img/"
    IMAGEURL "log/img/"
    QUERYFORMAT text/html
    LEGENDFORMAT text/html
    BROWSEFORMAT text/html

```

```

END

#... Layers includes files
INCLUDE "lays/dpa.lay"
INCLUDE "lays/sombra.lay"
INCLUDE "lays/relieve.lay"
INCLUDE "lays/caminos.lay"
INCLUDE "lays/costas.lay"
INCLUDE "lays/puentes.lay"

```

END

Obsérvese como se emplea en función de la incorporación de fuentes externas, el recurso denominado *fontset* y de igual forma para las simbologías *symbolset*. Por otra parte se pueden separar mediante el recurso *include* los distintos objetos que componen el mapa, tales como: *layer*, *web*, *outputformat*, *projection*, *legend*, *scalebar*, *querymap*, entre otros. Aunque evidentemente en el ejemplo que se expone durante el desarrollo de este acápite solo se excluyen las capas. En definitiva estas son las que se componen por un mayor cúmulo de información. Como bien se puede apreciar esta práctica minimiza la complejidad de mantenimiento del *mapfile*, teniendo en cuenta que de lo contrario el número de líneas crecería casi de forma exponencial.

A continuación se muestra el contenido de un fichero denominado *geolocal.font*, el cual almacena la declaración de las claves definidas para cada una de las fuentes que serán utilizadas en el mapa.

Vera	./fuentes/Vera.ttf
Arial	./fuentes/arial.ttf
Webdings	./fuentes/webdings.ttf
TwCenMTCondensed	./fuentes/TCCM.TTF
GillSansMTCondensed	./fuentes/GILC_.TTF
ARIALN	./fuentes/arial.ttf
ArialM	./fuentes/Arial.ttf
ArialNarrow	./fuentes/ARIALNB.TTF
SymbolosMM	./fuentes/Symbolos_MM.ttf

La simbología que será empleada por cada una de las estructuras que conforman el mapa también pueden ser descritas en un fichero independiente al *mapfile*, y de esta forma tener un control más preciso de este tipo de recursos, a continuación se muestra el contenido del fichero *geolocal.sym*, el contiene este tipo de información en función del proyecto en cuestión.

SYMBOLSET

```

SYMBOL
    name "circle"
    Type ELLIPSE
    Filled true
    POINTS 1 1 END

```

END

```

SYMBOL
    name "dashed"
    Type ELLIPSE
    FILLED true
    POINTS 1 1 END
    STYLE 5 3 END

```

END

```

SYMBOL
    NAME "oneway"
    TYPE TRUETYPE
    FONT scb
    CHARACTER "&#8594;"

```

```

        GAP -150
END

SYMBOL
  NAME "oneway"
  TYPE TRUETYPE
  FONT scb
  CHARACTER "&#8594;"
  GAP -150
END

```

Nótese como en el fragmento anterior se emplea la palabra reservada *symbolset* al inicio del documento. Siendo esa una especificación definida por Mapserver y en caso de no ser declarada induciría un error.

Como bien se había expresado en acápite anteriores, existe una amplia gama de formatos en que se puede almacenar el MDT. Entre los más estandarizados se encuentra el GML. Este presenta implementaciones en varias plataformas, incluso tiene soporte en algunas librerías de representación tales como el *OpenLayer*. A continuación se muestra el contenido del fichero denominado *caminos.lay*, el cual describe como se declara una capa con estas especificaciones.

```

LAYER
  CONNECTION "..../data/org/Cuba_Caminos.gml"
  CONNECTIONTYPE OGR
  NAME "caminos"
  STATUS DEFAULT
  TYPE POINT
  UNITS METERS
  CLASS
    STYLE
      ANGLE 360
      COLOR 200 11 2
      OUTLINECOLOR 50 90 0
      SYMBOL 0
      WIDTH 1
    END
  END
END

```

La información *raster* como bien se había mencionado en acápitres anteriores, potencia una forma visualización con un enfoque más realista del contenido cartográfico, además de los disimiles métodos analíticos que se soportan sobre este modelo. Por tanto su empleo es una realidad. Para mayor comprensión véase en el ejemplo que se muestra a continuación como se definen este tipo de capas.

```

LAYER
  NAME "sombra"
  GROUP 'Mapa_250000'
  STATUS ON
  TEMPLATE "ttt"
  TILEINDEX "..../data/org/Cuba_Shade.shp"
  TILEITEM "location"
  TRANSPARENCY 90
  UNITS DD
  EXTENT -88.870862 15.386740 -68.962302 26.758788
  TYPE RASTER

PROJECTION
  'proj=longlat'
  'ellps=WGS84'
  'atum=WGS84'
  'no_defs'
END

```

```

CLASS
  NAME "Sombra"
  EXPRESSION ([pixel] != 221)
  UNITS METERS
  STYLE
    COLORANGE 153 102 51 152 102 51
    DATARANGE 0 255
    RANGEITEM "[pixel]"
    COLOR 100 100 100
  END
END

```

Un elemento a destacar en el fragmento anterior es que las capas de tipo *raster* se visualizan a partir de un fichero *shape*, el cual se emplea generalmente como contenedor de información vectorial, siendo esto un elemento contradictorio. Esto se debe a que en este caso se emplea en función de almacenar la información asociada a la fuente real de datos. Teniendo en cuenta que este tipo de modelo se compone por un número ilimitado de ficheros de extensión: *hgt*, *tif*, etc. Por tanto se define la propiedad *tileindex* que define la localización del *shape*, así como *tileitem* con el objetivo de identificar cual es el atributo dentro del *shape*, que describe las rutas de acceso para cada uno de los elementos que componen el modelo.

4.2.3 Filtrado de datos y expresiones

Como bien se describe en acápite anteriores las capas son consideradas contenedores de información. Permitiendo describir como será visualizada la misma a través de clases y estilos. Sin embargo podemos enfrentarnos a situaciones en que el volumen de datos para una *layer* determinada sea muy grande y se requiera visualizar porciones de dicho contenido de forma diferente. En este caso los filtros y expresiones juegan un papel preponderante, permitiendo extraer a partir de un criterio determinado un subconjunto al cual se le podrá aplicar una amplia gama de estilos. A continuación se muestra el contenido definido para la capa denominada *dpa* como nomenclatura del término división política administrativa en la cual se emplea dicho recurso.

```

LAYER
  CONNECTION "../data/org/Cuba_Provincias.tab"
  CONNECTIONTYPE OGR
  NAME "dpa"
  STATUS DEFAULT
  TYPE POLYGON
  UNITS METERS

CLASS
  EXPRESSION ( "[provincia]" =~ /Cien/ )
  STYLE
    ANGLE 360
    COLOR 50 250 155
    OUTLINECOLOR 0 0 0
    SYMBOL 0
    WIDTH 1
  END
END

CLASS
  STYLE
    ANGLE 360
    COLOR 200 150 2
    OUTLINECOLOR 0 0 0
    SYMBOL 0
    WIDTH 1

```

```

        END
    END
END

```

Como se puede apreciar en el fragmento anterior se definen apenas dos etiquetas de tipo *class*. Una en la que se describe como debe de visualizarse el contenido del fichero ubicado en el directorio *data/org/* y denominado *Cuba_Provincias.tab*. En la otra clase se realiza un filtro sobre dicha fuente de datos, tomando como base el atributo *provincia*, especificándole la expresión regular que plantea una exclusión de los datos que contengan un valor similar a *Cien*, empleando para ello la propiedad *expression*.

4.3 MapScript

Históricamente el concepto del MapScript fue introducido en 2001 cuando la empresa canadiense DM Solutions desarrolló la API del MapServer para el lenguaje de programación PHP, en una extensión denominada PHP/MapScript. De manera sucinta, esta librería hace posible la disponibilidad de los recursos del MapServer para una amplia gama de lenguajes de programación, tales como *PHP*, *Pitón*, *Perl*, *Ruby*, *TCL*, *Java*, *C#*. De esa forma, se puede combinar los recursos del servidor de mapas con los de la tecnología requerida en función de su soporte, visando la creación de aplicaciones con un grado de personalización mayor, eventualmente no alcanzado con aplicaciones en modo CGI.

4.3.1 Cargando un mapa

El empleo de la Mapscript es bastante simple y presenta una analogía conceptual muy cercana a lo que se define en el Mapfile. A continuación se muestra un ejemplo básico que ilustra como cargar un mapa, para el cual se requieren al menos dos ficheros. El primero sería el que describe como debe ser confeccionada la imagen del mapa, denominado *mapfile* y de extensión *.map*, el cual no se muestra porque no es relevante en este caso, así que se asume cualquiera de los expuestos en acápitulos anteriores. El segundo hace referencia al *script* que se responsabiliza de crear un mapa utilizando de la librería Mapscript, el cual se denomina *basicmap.php* y su contenido sería el que se muestra a continuación.

```

<?php
if (extension_loaded("MapScript"))
    dl('php_mapscript.'.PHP_SHLIB_SUFFIX);
else die("Error: MapScript has not been installed");

$objMap = ms_newMapObj("mapfile.map");
$img = $objMap->draw();
$url = $img->saveWebImage();

?>

<html>
    <head>
        <title>Equipo #1 con Mapscript</title >
    </head>

    <body>
        <img src=<?php echo "./$url"; ?> >
    </body>
</html>

```

Nótese como este ejemplo se fracciona en dos partes fundamentales y que pueden diferenciarse fácilmente por el lenguaje utilizado. En la primera se emplea el *php* para

generar la imagen del mapa. Obsérvese como se utilizan las funciones nativas *extension_loaded* y *dl*, aunque estas no son realmente necesarias, sin embargo se recomienda su empleo pues garantizan la carga de la librería y se acota el margen de posibles errores. Posteriormente se utiliza la función *ms_newMapObj* para crear a partir del fichero denominado *mapfile.map* un objeto mapa, mediante el cual se invoca la construcción de la imagen a través del recurso *draw* y se le indica su almacenamiento con las propiedades especificadas en el **.map* mediante *saveWebImage*.

La segunda parte se define en lenguaje *HTML* en función de visualizar en el navegador la imagen generada del mapa solicitado. Para ello se define en el cuerpo de la página una etiqueta *img* especificándosele en la propiedad *src* la dirección que se obtuvo como resultado del recurso *saveWebImage*.

4.3.2 Adicionando elementos de interacción

En el acápite anterior se describió de forma muy básica como confeccionar un mapa utilizando la *Mapscript*. Sin embargo en no pocas ocasiones requerimos cierto nivel de interactividad, que nos permita desplazarnos, realizar análisis, consultas, entre otras cosas. Por tanto se requiere un esquema organizacional un tanto más complejo, a continuación se expone un ejemplo que garantiza la navegación y consumo de información.

Para llevar a cabo este proyecto se requiere definir algunos directorios que permitan un mayor nivel organizativo. Por tanto se crea una carpeta denominada *src* donde se encontrará todo el código fuente, así como las especificaciones del **.map*. En este caso se asume un *mapfile* que utiliza las especificaciones de capas de forma externa, las cuales se almacenarán en el directorio *src/lays/*. Además se independiza la creación de interfaces gráficas de la lógica para la gestión de mapas, dejándose solo el código *html* en el fichero *index.php* y creándose los ficheros *MS_Admin.php* y *Server.php* en *src/*.

Por otra parte se crea el directorio *log* en función de almacenar las imágenes que se generan por petición, así como los mensajes de error arrojados por el propio servidor de mapas. Por último en la carpeta *data* se encontrarán físicamente los datos cartográficos. Para mayor comprensión refiérase a la figura que se muestra a continuación.

Nombre	Tamaño	Tipo	Fecha de modificación
src	5 elementos	carpeta	vie 07 sep 2012 11:43:25 CDT
lays	7 elementos	carpeta	sáb 03 mar 2012 16:26:47 CST
sombra.lay	680 bytes	documento de texto sencillo	jue 01 mar 2012 11:20:03 CST
relieve.lay	1,5 KiB	documento de texto sencillo	jue 01 mar 2012 13:31:43 CST
puentes.lay	297 bytes	documento de texto sencillo	jue 01 mar 2012 09:27:55 CST
pueblos.lay	296 bytes	documento de texto sencillo	jue 01 mar 2012 09:26:50 CST
dpa.lay	485 bytes	documento de texto sencillo	sáb 03 mar 2012 16:26:47 CST
costas.lay	463 bytes	documento de texto sencillo	jue 01 mar 2012 09:27:29 CST
caminos.lay	296 bytes	documento de texto sencillo	sáb 03 mar 2012 16:26:26 CST
generated	4 elementos	carpeta	vie 07 sep 2012 11:43:25 CDT
Server.php	922 bytes	script en PHP	sáb 03 mar 2012 16:27:34 CST
MS_Admin.php	9,2 KiB	script en PHP	vie 11 may 2012 16:38:48 CDT
mapfile.map	1,4 KiB	documento de texto sencillo	sáb 03 mar 2012 16:30:29 CST
log	2 elementos	carpeta	mar 28 feb 2012 11:14:11 CST
data	1 elemento	carpeta	jue 14 jun 2012 17:17:44 CDT
org	22 elementos	carpeta	jue 01 mar 2012 12:15:47 CST
index.php	1,6 KiB	script en PHP	sáb 03 mar 2012 16:36:38 CST

Ilustración 13 Estructura del proyecto basado en la inclusión de fichero externos

Como bien se había expresado anteriormente el fichero *index.php* define la interfaz de cara al usuario, por tanto debe probar algunos recursos que garanticen la interactividad, en función de ganar encuanto a usabilidad del producto. Para ello se procede a crear en el cuerpo de la página una tabla que contenga el listado de las opciones del sistema. A continuación se expone el código que contiene el mismo.

```
<?php include "src/Server.php"; ?>
<html>
    <head>
        <TITLE>Mapscript Example</TITLE>
    </head>

    <body>
        <CENTER><FORM METHOD=POST ACTION="" >
            <TABLE>
                <TR>
                    <TD> <INPUT TYPE=IMAGE NAME="mapa" SRC="php
                        echo "./$URL"; ?&gt;"&gt;
                    &lt;/TD&gt;
                &lt;/TR&gt;
                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt; Pan: -----&lt;/b&gt; &lt;/TD&gt;
                    &lt;TD&gt; &lt;INPUT TYPE=RADIO name='action' value="0" CHECKED &gt; &lt;/TD&gt;
                &lt;/TR&gt;
                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Zoom In: -----&lt;/b&gt;&lt;/TD&gt;
                    &lt;TD&gt; &lt;INPUT TYPE=RADIO name='action' value="1" &gt; &lt;/TD&gt;
                &lt;/TR&gt;
                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Zoom Out: -----&lt;/b&gt;&lt;/TD&gt;
                    &lt;TD&gt; &lt;INPUT TYPE=RADIO name='action' value="-1" &gt; &lt;/TD&gt;
                &lt;/TR&gt;
                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Zoom Size: -----&lt;/b&gt;&lt;/TD&gt;
                    &lt;TD&gt; .....&lt;INPUT TYPE=TEXT name="factor" SIZE=2 value='3'&gt; &lt;/TD&gt;
                &lt;/TR&gt;
                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Full Extent: -----&lt;/b&gt;&lt;/TD&gt;
                    &lt;TD&gt; .....&lt;INPUT TYPE=SUBMIT name="full" value="Do" SIZE=2&gt; &lt;/TD&gt;
                &lt;/TR&gt;

                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Query:&lt;/b&gt; &lt;?php
                        if(isset($data)) echo '&lt;pre&gt;';
                        print_r( $data );
                        echo '&lt;/pre&gt;'&gt;
                    &lt;/TD&gt;
                    &lt;TD&gt; &lt;INPUT TYPE=RADIO name='action' value="q" &gt;
                        &lt;INPUT TYPE=TEXT name="query" value="puentes" SIZE=2&gt;
                    &lt;/TD&gt;
                &lt;/TR&gt;

                &lt;TR&gt;
                    &lt;TD&gt; &lt;b&gt;Information:&lt;/b&gt; &lt;?php
                        echo '&lt;pre&gt;';
                        print_r( $inf );
                        echo '&lt;/pre&gt;'&gt;
                    &lt;/TD&gt;
                &lt;/TR&gt;

            &lt;/TABLE&gt;
            &lt;INPUT TYPE=HIDDEN NAME="extent" VALUE='&lt;?php echo $extent ?&gt;'&gt;
        &lt;/FORM&gt;&lt;/CENTER&gt;
    &lt;/body&gt;
&lt;/html&gt;</pre

```

Nótese que a diferencia del ejemplo anterior el único código *php* que se incluye junto al *HTML* es el relacionado con la inclusión de *Server.php* y algunas validaciones propias de la *GUI*. Por otra parte a diferencia del *img* utilizado en el ejemplo anterior, se emplea la etiqueta *input* especificándose en la propiedad *type* el valor *image*. Este componente permite tener un mayor control sobre la imagen, manejando la información asociada a las coordenadas expresadas en pixeles, en función de la selección especificada por el usuario a través del puntero del *mouse*.

Por otra parte en la interfaz se muestra un componente *groupbox* que permite realizar una selección de carácter excluyente, entre las distintas opciones tales como:

- **Pan:** Permite realizar desplazamiento geográfico en cualquier dirección, para ello se basa en el mecanismo de recentrado del mapa, tomando como centro el punto seleccionado por el cliente a través del puntero del *mouse*.
- **Zoom In:** Permite realizar acercamientos sobre el mapa, tomando como factor el valor definido por el campo texo denominado *Zoom Size*, se activa con la acción del *clic* izquierdo del *mouse*.
- **Zoom Out:** Permite realizar alejamientos sobre el mapa, tomando como factor el valor definido por el campo texo denominado *Zoom Size*, se activa con la acción del *clic* izquierdo del *mouse*.
- **Full Extent:** Permite realizar ajustes de visualización al mapa en función de la máxima escala definida para el mismo, esta opción no es seleccionable y se ejecuta una vez accionado el botón denominado *Do*.
- **Query:** Permite realizar una consulta puntual, al ser seleccionada se debe especificar el nombre de la capa a consultar, se activa con la acción del *clic* izquierdo del *mouse* sobre el mapa.

Para mayor comprensión de lo anteriormente expuesto véase la figura que se muestra a continuación.

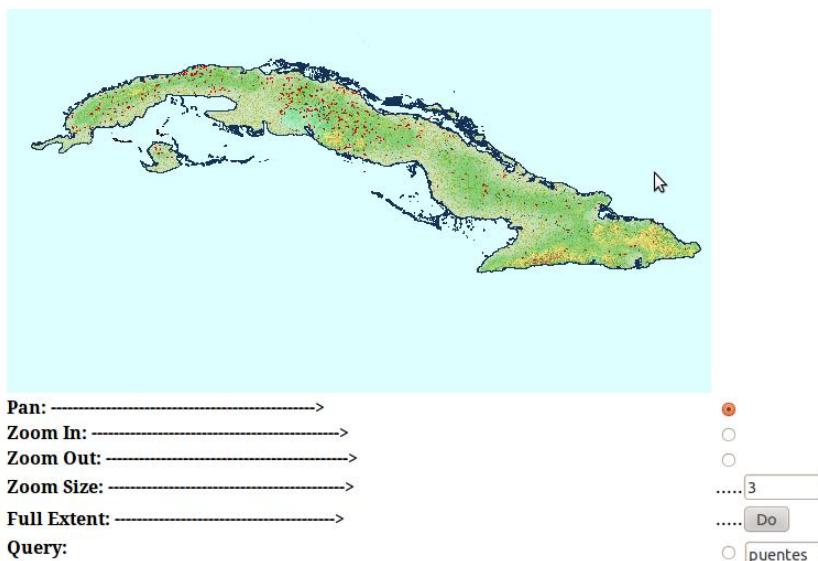


Ilustración 14 Vista del ejemplo2, basado en la generación de mapas estáticos

En función de suplir el código definido en el ejemplo anterior, responsabilizado de la construcción del mapa, se define el script denominado *Server.php* como controlador frontal. Este decepciona las peticiones del cliente, teniendo en cuenta que no solo se visualiza el mapa, sino que también se pueden realizar diferentes acciones de navegación como acercamientos, alejamientos, desplazamientos en distintas direcciones y consumo

de información puntual.

El *extent* del mapa se almacena en una etiqueta oculta en formato *json*, en función de minimizar la complejidad de su posterior interpretación en el servidor. Las coordenadas de pantalla asociadas a la selección del mouse pueden ser accedidas a través de las claves [*mapa_x*, *mapa_y*] de la variable global *\$_REQUEST*. Por otra parte cada una de las acciones esta compuesta por un identificador numérico, en caso de [*Pan*, *Zoom In*, *Zoom Out*, *Query*] pueden ser accedidas a través de la clave *action*, el *Full Extent* a través de *full* y el identificador de la capa mediante *query*.

En caso de que la acción sea de consulta, se procede a convertir las coordenadas de pantalla a geográficas mediante el recurso *MsAdmin::pix2Geo*, y a través de la función *MsAdmin::queryByPoint* se puede obtener dicha información, contrario a esto se procede a invocar la función *MsAdmin::navigate*, garantizando de esta forma los procesos de navegación.

```
<?php
    include "Ms_Admin.php";
    //... static map
    $msAdm = new MsAdmin(true, "src/mapfile.map");
    //... in controls ...
    if ( !isset($_REQUEST["full"]) ) && isset($_REQUEST["action"]))
    {
        $extent = json_decode($_REQUEST["extent"], true);
        $center = isset($_REQUEST["mapa_x"])?array( $_REQUEST["mapa_x"], $_REQUEST["mapa_y"])
    ):0;

        if($_REQUEST["action"] == 'q')
        {
            $x = $msAdm->pix2Geo($center[0], 0, 640, $extent['minx'], $extent['maxx']);
            $y = $msAdm->pix2Geo($center[1], 0, 480, $extent['miny'], $extent['maxy']);
            $data = $msAdm->queryByPoint($x, $y, $_REQUEST["query"]);
        }

        $msAdm->navigate($_REQUEST["action"], $_REQUEST["factor"], $center, $extent);
        $extent = $msAdm->getExtentAsJson();
    }
    }else if(isset($_REQUEST["extent"])) $extent = $_REQUEST["extent"];
    $inf = $msAdm->getinfo();
    //... out controls ...
    $URL = $msAdm->draw();
?>
```

Obsérvese que en el fragmento de código anterior, el tratamiento del mapa no se realiza de forma directa, sino que se delega a la clase denominada *MsAdmin* la cual se encuentra descrita en el fichero *Ms_Admin.php*. Esto permite encapsular los comportamientos asociados a la administración del servidor de mapas, la cual a medida que se avance en la lectura de este capítulo irá evolucionando, sin necesidad de modificar el resto de los elementos que componen este proyecto que se toma de ejemplo.

```
<?php
/*
 * @description: Clase para la administración de mapas sobre Mapserver
 * @copyright: T0N3 KSK
 */
class MsAdmin
{
    //.....Declaracion de variables.....
    protected $objMap;
    public $imagspath;
    public $imagurl;
    public $maptmp;
```

```

//.....Declaración de métodos.....
public function __construct($auto=true, $mapfile=0)
{
    if (!extension_loaded("MapScript"))
        dl('php_mapscript.'.PHP_SHLIB_SUFFIX);

    $this->pathc = dirname(__FILE__) . '/..';
    $this->cache = 5;
    $this->logpath = 'log/error/trace.log';
    if($auto) $this->makeMap($mapfile);
}

public function makeMap($mapfile=0)
{
    $path = $this->path . 'data/org/';
    //... Limpiar lista de errores del servidor de mapas
    $this->clear();
    // Construir el objeto mapa sin fichero de configuracion
    $this->buildmap($mapfile);
    //... Controla la salida de errores
    $this->showError();
    //... Rertornar Objeto Mapa
    return $this->objMap;
}

public function buildmap($mapfile=0)
{
    $this->objMap = ms_newMapObj($mapfile);
}

public function showError()
{
    $error = ms_GetErrorObj();
    $lserr = array();
    while($error)
    {
        if(!empty($error->message)) {
            $out = sprintf("Mapscript Error >> %s: %s \r\n",
                $error->routine,
                $error->message
            );
            file_put_contents($this->logpath, $out);
            $lserr[] = $out;
        }
        $error = $error->next();
    }
    return $lserr;
}

public function clear()
{
    ms_ResetErrorList();
    $dir = $this->path. 'log/img/';
    $count = 0;
    if ($gestor = opendir($dir)) {
        while (false !== ($archivo = readdir($gestor)))
            if ($archivo != "." && $archivo != "..") $count++;
        closedir($gestor);
    }
    if($count > $this->cache) exec(" rm -r $dir* & chmod -R 777 $dir ");
}

public function navigate($mode=0, $factor=1, $center=0, $ext=0)
{
    $factor = $factor ? $factor : 1;
    $ext = $ext ? $ext : (array) $this->objMap->extent;
    $center = $center ? $center : array(
        ($ext['maxx'] - $ext['minx'])/2,
        ($ext['maxy'] - $ext['miny'])/2
    );
    $this->objMap->setextent($ext['minx'], $ext['miny'], $ext['maxx'], $ext['maxy']);
}

```

```

$point = ms_newpointObj();
$point->setXY($center[0], $center[1]);
$extent = ms_newrectObj();
$extent->setextent($ext['minx'], $ext['miny'], $ext['maxx'], $ext['maxy']);
$factor = $mode!=0 ? $mode * $factor : 1;
$this->objMap->zoompoint(
    $factor,
    $point,
    $this->objMap->width,
    $this->objMap->height,
    $extent
);
}

public function getExtentAsJson($extent=0)
{
    $extent = (!$extent) ? (array) $this->objMap->extent : $extent;
    if(isset($extent['_handle_'])) unset($extent['_handle_']);
    return json_encode($extent);
}

public function draw()
{
    $imagen = $this->objMap->draw();
    return $imagen->saveWebImage();
}
/*
 * converts pixel coordinates to geo
 * @param pixPos coordinate in pixel
 * @param pixMin minimum pix coordinate
 * @param pixMax maximum pix coordinate
 * @param geoMin minimum geo coordinate
 * @param geoMax maximum geo coordinate
 */
public function pix2Geo($pixPos, $pixMin, $pixMax, $geoMin, $geoMax) {
    return $geoMin + ($pixPos - $pixMin) * ($geoMax - $geoMin) / ($pixMax - $pixMin);
}
/*
 * converts geo coordinates to pix
 * @param geoPos coordinate in geo
 * @param pixMin minimum pix coordinate
 * @param pixMax maximum pix coordinate
 * @param geoMin minimum geo coordinate
 * @param geoMax maximum geo coordinate
 */
public function geo2Pix($geoPos, $geoMin, $geoMax, $pixMin, $pixMax) {
    return $pixMin + ($geoPos - $geoMin) * ($pixMax - $pixMin) / ($geoMax - $geoMin);
}

public function queryByPoint($X, $Y, $layername='contorno')
{
    ms_ResetErrorList();
    $X = $X ? $X : ( $this->objMap->extent->maxx - $this->objMap->extent->minx ) /2;
    $Y = $Y ? $Y : ( $this->objMap->extent->maxy - $this->objMap->extent->miny ) /2;
    $layername = $layername ? $layername : 'contorno';
    $msPoint = ms_newPointObj();
    $msPoint->setXY($X, $Y);

    if($msLayer = $this->objMap->getLayerByName($layername))
    {
        $status = $msLayer->status;
        $maxscaledenom = $msLayer->maxscaledenom;
        $minscaledenom = $msLayer->minscaledenom;
        $msLayer->set('maxscaledenom', -1);
        $msLayer->set('minscaledenom', -1);
        $msLayer->set('status', MS_ON);

        $this->objMap->preparequery();
        $qresult = @$msLayer->queryByPoint($msPoint, MS_SINGLE, -1);
        if ($qresult == MS_SUCCESS)

```

```

    {
        $msLayer->open();
        $nresult = $msLayer->getNumResults();
        if ($nresult)
        {
            $xresult = $msLayer->getResult(0);
            $result = $msLayer->getShape(
                $xresult->tileindex,
                $xresult->shapeindex
            );
            $out    = $result->values;
            $result->free();
        }
        $msLayer->close();
    }
    ms_ResetErrorList();
    $msLayer->set('maxscaledenom', $maxscaledenom);
    $msLayer->set('minscaledenom', $minscaledenom);
    $msLayer->set('status', $status);
    return isset($out) ? $out : 'the source is empty or it is bat query';
} else return false;
}

public function getInfo()
{
    return array(
        'name' => $this->objMap->name,
        'status' => $this->objMap->status,
        'width' => $this->objMap->width,
        'height' => $this->objMap->height,
        'projection' => $this->objMap->getProjection(),
        'extent' => (array) $this->objMap->extent,
        'layers' => $this->objMap->getAllLayerNames()
    );
}
}

```

Como bien se puede observar esta clase implementa una serie de funciones que facilitan la gestión de mapas a través de la Mapscript. A continuación se describen los principales elementos que la componen.

- **__construct:** El constructor de la clase básicamente se encarga de garantizar la carga de la librería Mapscript, además de la inicialización de aquellas propiedades de carácter global tales como: *pathc* (directorio relativo a la raíz del proyecto), *cache* (valor máximo de ficheros temporales almacenados en disco), *logpath* (directorio en el cual se almacenaran en forma de *log* los mensajes de error). Recibe por parámetro dos valores, el primero hace referencia a la carga automática del mapa invocando para ello la función *MsAdmin::makeMap*, *tomando valor verdadero por defecto*, y el segundo parámetro se refiere a la dirección física del *mapfile*.
- **makeMap:** Define el procedimiento de creación del mapa, en el cual se debe verificar el limpiado de la cache, invocar la construcción y registrar los errores detectados.
- **buildmap:** Encapsula la construcción del mapa, y aunque en este caso es muy simple permitirá sentar las bases para proyectos de mayor complejidad.
- **showError:** Se responsabiliza de detectar los errores arrojados por el servidor de mapas a través del recurso *ms_GetErrorObj*, y almacenarlos en forma de trazas o *log*.
- **clear:** Procede a limpiar la lista de errores del *Mapserver* a través de la función *ms_ResetErrorList*, además de chequear que el número de ficheros temporales

almacenados en disco como las imágenes generadas del mapa, no exceda el límite definido por la variable global denominada *cache*, en caso positivo intentaría eliminarlos automáticamente.

- **navigate:** Garantiza los procesos de navegación territorial, para ello emplea la función *zoompoint* implícita en el objeto mapa, la cual requiere un factor de desplazamiento, el punto que será tomado como centro, creándose a través de la función *ms_newpointObj*, además de las dimensiones del mapa a través de las propiedades [*width*, *height*], así como el *extent* de visualización, estructura que se obtiene a través de la función *ms_newrectObj*, especificándosele dos puntos extremos [(*minx*, *miny*) (*maxx*, *maxy*)]. Basicamente cuando la acción se refiere al comportamiento de *Pan*, el *extent* no varia, únicamente cambia el valor del centro dando de esta forma la sensación de movimiento.
- **getExtentAsJson:** Esta función transforma la estructura almacenada en la propiedad *extent* implícita en el objeto mapa, al formato *json*.
- **draw:** Encapsula el comportamiento de creación de la imagen y almacenamiento en disco, equivalente a la ejecución sincrónica de las funciones *draw* y *saveWebImage*.
- **pix2Geo:** Transforma una componente de un punto en coordenadas de pantalla o pixel a geográfica, para ello requiere el valor a convertir, así como la correlación de extensión territorial con las dimensiones de la imagen.
- **geo2Pix:** Transforma una componente de un punto en coordenadas geográficas a *pixels*.
- **queryByPoint:** Permite consultar la información asociada a una capa determinada, tomando como base un punto referenciado geográficamente. Para ello procede a buscar la instancia de la capa por el nombre, almacena sus propiedades básicas y la habilita, de lo contrario no se podría efectuar la consulta. Luego invoca la función *queryByPoint* implícita en la instancia de la *layer* correspondiente, especificándosele el punto de referencia conformado mediante *ms_newPointObj*, así como el rango de selección, una vez obteniendo el resultado se procede a evaluar su contenido transformando la estructura de tipo *Shape* que se obtiene a partir de la función *getShape*, y finalmente se librera la memoria ocupada, se limpia la lista de errores del Mapserver y se restablecen los valores iniciales de la *layer*.
- **getinfo:** Retorna un arreglo asociativo con información referente al mapa basándose en las siguientes claves [*name* / *status* / *width* / *height* / *projection* / *extent* / *layers*]

4.3.3 Confeccionando un mapa dinámico

Hasta el momento se han visto los mecanismos de creación de mapas digitales mediante la *Mapscript*, tomando como referencia un fichero que describe toda la configuración suficiente y necesaria que le permita al *Mapserver* arrojar una salida. También es posible confeccionarlos sin necesidad de la existencia de este fichero físico, incorporándole las propiedades dinámicamente al *mapObj*¹⁷.

Esto podría ser útil en caso de un sistema que requiera el control de perfiles cartográficos por usuarios. En principio podría pensarse en una solución que almacene en un directorio del servidor los distintos *mapfile* por cliente. Sin embargo esto se vería influenciado por el entorno de despliegue y el número de personas que utilicen dicho software. Por tanto esta información podría crecer tanto y ser tan compleja, al punto que la gestión de la misma

¹⁷ **mapObj** también se le conoce como objeto mapa, es una instancia obtenida a partir de la invocación de *ms_newMapObj(\$mapfile)* la cual permite tener control total sobre el mapa digital gestionado con Mapserver.

llevaría al producto ser prácticamente inusuable. Quizás la mejor variante sería almacenar toda esa información en un servidor de bases de datos y construir el mapa de forma dinámica.

En función de solventar dicho planteamiento se toma como base el proyecto que confecciono en el acápite anterior, realizándole modificaciones puntuales que permitan obtener el resultado esperado. Por tanto la estructura física se mantiene muy similar, únicamente se elimina el *mapfile.map* y el directorio *src/lays/* pues estos no serán utilizados, para mayor comprensión véase la figura que se muestra a continuación.

Nombre	Tamaño	Tipo	Fecha de modificación
src	3 elementos	carpeta	vie 07 sep 2012 11:43:25 CDT
generated	4 elementos	carpeta	vie 07 sep 2012 15:45:36 CDT
server.php	918 bytes	script en PHP	vie 07 sep 2012 15:48:40 CDT
MsAdmin.php	9,2 KiB	script en PHP	vie 11 may 2012 16:38:48 CDT
log	2 elementos	carpeta	mar 28 feb 2012 11:14:11 CST
img	3 elementos	carpeta	vie 07 sep 2012 15:41:48 CDT
504a4dfc_5d8_0.png	84,2 KiB	imagen PNG	vie 07 sep 2012 15:41:48 CDT
504a4d88_5db_0.png	84,2 KiB	imagen PNG	vie 07 sep 2012 15:39:52 CDT
4fb51290_9b0_0.png	84,7 KiB	imagen PNG	jue 17 may 2012 11:00:32 CDT
error	1 elemento	carpeta	vie 02 mar 2012 15:07:29 CST
trace.log	67 bytes	registro de aplicación	sáb 03 mar 2012 16:29:33 CST
data	1 elemento	carpeta	jue 14 jun 2012 17:17:44 CDT
org	22 elementos	carpeta	jue 01 mar 2012 12:15:47 CST
index.php	1,6 KiB	script en PHP	vie 07 sep 2012 15:41:43 CDT

Ilustración 15 Estructura física del ejemplo para la creación de un mapa dinámico

Lo primero que debe ser modificado es el controlador frontal denominado *server.php* almacenado en el directorio *src/*. En el cual se sustituiría la línea que se construye la instancia de la clase *MsAdmin*, pues en este caso no se estará utilizando un fichero de configuración de mapa, quedando de la siguiente forma:

```
$msAdm = new MsAdmin();
```

El resto del código fuente se mantiene idéntico para el controlador frontal. Nótese que en este caso se asumen los parámetros por omisión, por tanto estos toman los valores definidos por defecto. A continuación se muestra las modificaciones del código fuente perteneciente a la clase *MsAdmin*, en caso de no sufrir modificaciones algún método se empleara la notación de los tres puntos suspensivos (...)

```
<?php
/*
 * @description: Clase para la administración de mapas sobre Mapserver
 * @copyright: T0N3 KSK
 */
class MsAdmin
{
    //.....Declaración de variables.....
    protected $objMap;
    public $imagpath;
    public $imagurl;
    public $maptmp;
    //.....Declaración de métodos.....
    public function __construct($auto=true, $mapfile=0)
    {
        if (!extension_loaded("MapScript"))
            dl('php_mapscript.'.PHP_SHLIB_SUFFIX);
```

```

    $this->pathc = dirname(__FILE__) . '/..';
    $this->cache = 5;
    $this->imagpath = "log/img/";
    $this->imagurl = "log/img/";
    $this->logpath = 'log/error/trace.log';
    if($auto) $this->makeMap($mapfile);
}

public function makeMap($mapfile=0)
{
    $path = $this->path . 'data/org/';
    //... Limpiar lista de errores del servidor de mapas
    $this->clear();
    //... Construir el objeto mapa sin fichero de configuración
    $this->buildmap($mapfile);
    //... Definición de Capas y sus propiedades
    $layerOptions = include $this->path.'src/generated/style.lays.php';
    $this->addLayer("caminos", $path."Cuba_Caminos.gml", $layerOptions['caminos']);
    $this->addLayer("puentes", $path."Cuba_Puentes.shp", $layerOptions['puentes']);
    $this->addLayer("costas", $path."Cuba_Costas.dxf", $layerOptions['costas']);
    $this->addLayer("pueblos", $path."Cuba_Pueblos.json", $layerOptions['pueblos']);
    //... Controla la salida de errores
    $this->showError();
    //... Rertornar Objeto Mapa
    return $this->objMap;
}

public function buildmap($mapfile=0)
{
    if(!$mapfile){
        $this->objMap = ms_newMapObj("");
        //... Propiedades
        $this->objMap->set("name","Cuba");
        $this->objMap->set("status", MS_ON);
        $this->objMap->imagecolor->setRGB(0, 150, 0);
        $this->objMap->setSize(640, 480);
        $this->objMap->setExtent(-85, 17.7331838565022, -74, 26.2668161434978);
        $this->objMap->setProjection( "init=epsg:4267" );
        $this->objMap->imagecolor->setRGB(0, 0, 56);
        $this->objMap->selectOutputFormat("PNG24");
        //... Definición del objeto Web
        $this->objMap->web->set( "imagepath", $this->imagpath );
        $this->objMap->web->set( "imageurl", $this->imagurl );
    }
    else $this->objMap = ms_newMapObj($mapfile);
}

public function addLayer($name, $source, $sld=array())
{
    $sld = $this->formatStyle($sld);

    $objLayer = ms_newLayerObj($this->objMap);
    $objLayer->set( "name", $name );
    $objLayer->set( "type", $sld['type'] );
    $objLayer->set( "status", MS_DEFAULT );
    $objLayer->set( "connection", $source );
    $objLayer->setConnectionType($sld['connectionType']);

    $objClass = ms_newClassObj($objLayer);
    $objStyle = ms_newStyleObj($objClass);

    $objStyle->color->setRGB(
        $sld['color']['R'],
        $sld['color']['G'],
        $sld['color']['B']
    );

    $objStyle->outlinecolor->setRGB(
        $sld['outlinecolor']['R'],
        $sld['outlinecolor']['G'],
        $sld['outlinecolor']['B']
    );
}

```

```

        $sld['outlinecolor']['B']
    );
}

private function formatStyle($sld)
{
    if(!isset($sld['outlinecolor']))
        $sld['outlinecolor'] = array('G'=> 20, 'R'=> 23, 'B'=> 10);
    if(!isset($sld['color']))
        $sld['color'] = array('G'=> 50, 'R'=> 23, 'B'=>90);
    if(!isset($sld['type']))
        $sld['type'] = MS_LAYER_POLYGON;
    if(!isset($sld['connectionType']))
        $sld['connectionType'] = MS_OGR;
    return $sld;
}
public function showError() { ... }
public function clear() { ... }
public function draw() { ... }
public function navigate($mode=0, $factor=1, $center=0, $ext=0) { ... }
public function getExtentAsJson($extent=0) { ... }
public function pix2Geo($pixPos, $pixMin, $pixMax, $geoMin, $geoMax) { ... }
public function geo2Pix($geoPos, $geoMin, $geoMax, $pixMin, $pixMax) { ... }
public function queryByPoint($X, $Y, $layername='contorno') { ... }
public function getInfo() { ... }
}
?>

```

El constructor no sufrió muchos cambios realmente, tan solo se le adicionaron algunas variables que antes se almacenaban en el *mapfile*, tales como *imagpath* la cual define el directorio en que se almacenaran la imágenes generadas, tomando por defecto el valor de *log/img/*, y la propiedad *imagurl* que describe la *url* de publicación tomando por defecto el valor de *log/img/*

Obsérvese como cambia el método *MsAdmin::buildmap*, ajustándose al modo de construcción del *mapObj* en función de los parámetros especificados. Pues en caso de que el *mapfile* no sea especificado o tome explícitamente valor 0, intentaría crear un objeto mapa totalmente dinámico y el mecanismo para ello es especificar el parámetro vacío para la función *ms_newMapObj("")*. Quizás este método podría implementarse un tanto mas genérico, valorando la opción de especificar la configuración del *mapfile* en modo texto, lo cual podría realizarse mediante la función *ms_newMapObjFromString*, o incluso crearlo como copia o clon de uno existente a través del empleo del segundo parámetro de *ms_newMapObj* o simplemente utilizando el recurso *clone*.

Un elemento importante a destacar en la construcción del *mapObj* de forma dinámica, es que al igual que otras estructuras propias de la Mapscript. Muchas de sus propiedades se especifican a través de la función *set*, recibiendo por parámetro el identificador semántico del atributo y su correspondiente valor. Sin embargo no todas cumplen con esta sintaxis, definiéndose métodos específicos para cada una de estas, tales como *setSize*, *setExtent*, *setProjection*, *selectOutputFormat*, entre otras.

Por otra parte en el método *makeMap* una vez instanciado el *mapObj*, se adiciona el proceso de incorporación de capas. Para el cual se define un método genérico, que se basa en la especificación de la configuración particular de cada *layer* que se desee incorporar, en este caso se introducen cuatro capas en distintos formatos, tales como: *gml*, *shp*, *dxf*, *json*.

La configuración anteriormente mencionada se almacena en un fichero denominado *style.lays.php* que se encuentra almacenado en el directorio *src/generated/*, cuyo

contenido se muestra a continuación.

```
<?php
$layerOptions['caminos']['color']['R'] = 20;
$layerOptions['caminos']['color']['G'] = 120;
$layerOptions['caminos']['color']['B'] = 20;
$layerOptions['caminos']['outlinecolor']['R'] = 1;
$layerOptions['caminos']['outlinecolor']['G'] = 3;
$layerOptions['caminos']['outlinecolor']['B'] = 4;
$layerOptions['caminos']['type'] = MS_LAYER_LINE;
$layerOptions['puentes']['color']['R'] = 200;
$layerOptions['puentes']['color']['G'] = 11;
$layerOptions['puentes']['color']['B'] = 2;
$layerOptions['puentes']['type'] = MS_LAYER_LINE;
$layerOptions['costas']['type'] = MS_LAYER_POINT;
$layerOptions['pueblos']['color']['R'] = 150;
$layerOptions['pueblos']['color']['G'] = 250;
$layerOptions['pueblos']['color']['B'] = 150;
return $layerOptions;
?>
```

Obsérvese que la estructura del fichero *style.lays.php* se basa en la especificación estándar de *php*, en el cual se define un arreglo asociativo, que almacena información indexada por el nombre de la capa. La cual se puede obtener a través de una simple llamada a la función nativa del propio lenguaje denominada *include*. También es importante destacar que esta información puede ser almacenada en bases de datos y para nada entra en contradicción con el objetivo planteado al principio, tan sólo se utilizó esta vía por su facilidad de implementación.

Si se procede a analizar a fondo el comportamiento de *MsAdmin::addLayer*, podrá identificarse que lo primero que se realiza, consiste en la invocación del método *MsAdmin::formatStyle*, este se encarga de rellenar los tributos que no sean especificados, pasándoles valores definidos por defecto. Por otra parte para adicionar una capa tanto se requiere la llamada de *ms_newLayerObj(MapObj map [, layerObj layer])*, a la cual se le debe especificar el *mapObj* y de forma opcional una instancia de tipo *LayerObj* en caso que se requiera clonar las propiedades de alguna existente.

Nótese como que para que se pueda observar el contenido de una capa además de estar activa, debe estar compuesta por al menos una clase que contenga un estilo bien definido. Para crear una clase se emplea el recurso *ms_newClassObj(layerObj layer [, classObj class])*, de igual forma el primer parámetro indica a la clase que será añadida y el segundo la instancia de aquella que asumirá el valor de sus propiedades. Por ultimo el comportamiento de instanciación del objeto estilo es similar al de sus contenedores: *ms_newStyleObj(classObj class [, styleObj style])*

4.3 SLD

En el acápite anterior se describe como adicionar elementos de forma dinámica a un mapa a través de la *Mapscript*. De la misma forma en el proceso de creación de capas, se define un mecanismo que permitía desde un fichero de configuración externo gestionar dichas propiedades. Sin embargo esto no resulta del todo necesario pues *Mapserver* provee un recurso que abstrae a los desarrolladores en este sentido, el cual se basa en el empleo de los *SLD*¹⁸.

Estos son considerados un esquema XML propuesto por *Open Geospatial Consortium*,

¹⁸ **SLD** acrónimo de *Styled Layer Descriptor*

como lenguaje estándar para describir los atributos del conjunto de capas que detallan la apariencia a un mapa. En los sistemas de información geográfico y servidores cartográficos permiten especificar el estilo visual de cada capa de objetos que componen el mapa, permitiendo, por ejemplo, representar el color de relleno, tipo y ancho de borde, etc.

En función de poner en práctica lo anteriormente mencionado se procederá a realizar algunas modificaciones al proyecto definido en el acápite anterior. En la práctica es muy común que se combinen las dos técnicas de construcción de mapas, dígase dinámica o estática, pues resultan beneficiosas bajo determinadas condiciones propias del entorno de negocio. Además se procederá a describir como generar un *mapfile* a partir de un *mapObj*, aplicar estilos basados SLD y generarlos a partir de un *mapObj*.

Nombre	Tamaño	Tipo	Fecha de modificación
src	5 elementos carpeta		vie 07 sep 2012 16:57:48 CDT
lays	7 elementos carpeta		sáb 03 mar 2012 16:26:47 CST
generated	4 elementos carpeta		vie 07 sep 2012 16:58:03 CDT
style.sld.xml	2,3 KiB	Microsoft Help Virtual Topic Definition File	sáb 03 mar 2012 16:25:43 CST
style.lays.php	759 bytes	script en PHP	jue 01 mar 2012 15:01:17 CST
mapfile.map	1,4 KiB	documento de texto sencillo	sáb 03 mar 2012 16:30:29 CST
lay.sld.xml	1,4 KiB	Microsoft Help Virtual Topic Definition File	sáb 03 mar 2012 16:25:37 CST
server.php	918 bytes	script en PHP	vie 07 sep 2012 15:48:40 CDT
MsAdmin.php	9,2 KiB	script en PHP	vie 07 sep 2012 15:58:42 CDT
mapfile.map	1,4 KiB	documento de texto sencillo	sáb 03 mar 2012 16:30:29 CST
log	2 elementos carpeta		mar 28 feb 2012 11:14:11 CST
data	1 elemento carpeta		jue 14 jun 2012 17:17:44 CDT
index.php	1,6 KiB	script en PHP	vie 07 sep 2012 15:53:25 CDT

Ilustración 16 Taxonomía del ejemplo para la generación y carga de SLD

Como bien se puede observar en la imagen anterior, la estructura de este proyecto es idéntica a los anteriores. En este caso se comienza a almacenar en el directorio *src/generated/* los ficheros que son generados a través de la propia librería. A continuación se describen las modificaciones a la clase *MsAdmin*.

```
<?php
/*
 * @description: Clase para la administración de mapas sobre Mapserver
 * @copyright: T0N3 KSK
 */
class MsAdmin
{
    //.....Declaración de variables.....
    protected $objMap;
    public $imagpath;
    public $imagurl;
    public $maptmp;
    //.....Declaración de métodos.....
    public function __construct($auto=true, $mapfile=0)
    {
        if (!extension_loaded("MapScript"))
            dl('php_mapscript.'.PHP_SHLIB_SUFFIX);

        $this->pathc = dirname(__FILE__) . '/..';
        $this->cache = 5;
        $this->imagpath = "log/img/";
        $this->imagurl = "log/img/";
        $this->logpath = 'log/error/trace.log';
        $this->maptmp = 'src/generated/mapfile.map';
        $this->sldtmp = 'src/generated/lay.sld.xml';

        if($auto) $this->makeMap($mapfile);
    }
}
```

```

}

public function makeMap($mapfile=0)
{
    $path = $this->path . 'data/org/';
    //... Limpiar lista de errores del servidor de mapas
    $this->clear();
    // Construir el objeto mapa sin fichero de configuracion
    $this->buildmap($mapfile);
    //... Definición de Capas y sus propiedades
    $this->addLayerContor( $path."Cuba_Provincias.tab" );
    //... Controla la salida de errores
    $this->showError();
    //... Generar una salva temporal del mapa
    $this->objMap->save($this->path.$this->maptmp);
    //... Generar una salva temporal de los estilos del mapa
    $sld = $this->objMap->generateSLD();
    file_put_contents($this->path.$this->sldtmp, $sld);
    //... Aplicar un estilo a mapa desde un sld
    $sld = file_get_contents($this->path.$this->sldtmp);
    $this->objMap->applySLD($sld);
    //... Rertornar Objeto Mapa
    return $this->objMap;
}

private function addLayerContor ($source)
{
    $objLayer = ms_newLayerObj($this->objMap);
    $objLayer->set( "name", "dpa" );
    $objLayer->set( "type", MS_LAYER_POLYGON );
    $objLayer->set( "status", MS_DEFAULT );
    $objLayer->set( "connection", $source );
    $objLayer->setConnectionType(MS_OGR);

    //...Definición de la Clase 2 para la Capa
    $objClass2 = ms_newClassObj($objLayer);
    //... Definición expresiones regulares o filtros por Clases
    $expresion = "( \\" [provincia] \\" =~ /Cien/ )";
    $objClass2->setExpression($expresion);
    //... Definición de Etilos para la Clase 2
    $objStyle2 = ms_newStyleObj($objClass2);
    $objStyle2->color->setRGB(50, 250, 155);
    $objStyle2->outlinecolor->setRGB(0, 0, 0);

    //... Definición de la Clase 1 para la Capa
    $objClass = ms_newClassObj($objLayer);
    //... Definición de Etilos para la Clase
    $objStyle = ms_newStyleObj($objClass);
    $objStyle->color->setRGB(200, 150, 2);
    $objStyle->outlinecolor->setRGB(0, 0, 0);

    $laysId = $objLayer->generateSLD();
    file_put_contents($this->path.'src/ generated/lay.sld.xml', $laysId);
    $laysId = file_get_contents($this->path.'s');
    $objLayer->applySLD($laysId);
}

public function buildmap($mapfile=0) { ... }
public function showError() { ... }
public function clear() { ... }
public function draw() { ... }
public function navigate($mode=0, $factor=1, $center=0, $ext=0) { ... }
public function getExtentAsJson($extent=0) { ... }
public function pix2Geo($pixPos, $pixMin, $pixMax, $geoMin, $geoMax) { ... }
public function geo2Pix($geoPos, $geoMin, $geoMax, $pixMin, $pixMax) { ... }
public function queryByPoint($X, $Y, $layername='contorno') { ... }
public function getInfo() { ... }

}

?>

```

En este ejemplo se eliminan las funciones *addLayer* y *formatStyle*. Además se incorporan dos nuevas variables al constructor (*maptmp* y *sldtmp*), las cuales describen los directorios en que serán almacenadas a partir del *mapObj*, siendo el caso del *mapfile* y el *sld* respectivamente.

Obsérvese como la función *MsAdmin::makeMap* es modificada en función de lograr los objetivos planteados al inicio de este acápite. Para la generación automática del *mapfile* se emplea el recurso *sabe* implícito en el *mapObj*, al cual se le especifica la dirección donde será almacenado. De igual forma se invoca el método *generateSLD*, para obtener una cadena de texto basa en las especificaciones del *sld* a partir de las propiedades definidas en el objeto mapa, la cual se puede almacenar a través de la función nativa denominada *file_put_contents*.

Los fichero *sld* pueden ser aplicados tanto al objeto mapa, como a las instancias de las capas a través de la función *applySLD*, especificándole su contenido en forma de cadena de texto por parámetro. Previamente se debe haber obtenido dicha información mediante recursos nativos del lenguaje, tales como *file_get_contents*. La puesta en práctica de esta afirmación se puede observar en las funciones *MsAdmin::makeMap* y *MsAdmin::addLayerContor*. Esta última describe como se adiciona una capa a la cual se le especifica un filtro de forma dinámica, invocando para ello el comportamiento de *setExpression*.

Pro otra parte el fichero *style.sld.xml*, almacenado en el directorio *src/generated/*, contiene la información necesaria para describir como una capa determinada debe visualizarse. A continuación se muestra su contenido.

```
<StyledLayerDescriptor xmlns:ogc="http://www.opengis.net/ogc" xmlns="http://www.opengis.net/sld"
  xmlns:gml="http://www.opengis.net/gml" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sld
  http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>dpa</Name>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <ogc:Filter>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>provincia</ogc:PropertyName>
              <ogc:Literal>Cienfuegos</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Filter>

          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#32fa9b</CssParameter>
              <CssParameter name="fill-opacity">1.00</CssParameter>
            </Fill>
            <Stroke>
              <CssParameter name="stroke">#000000</CssParameter>
              <CssParameter name="stroke-width">1.00</CssParameter>
            </Stroke>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#c89602</CssParameter>
              <CssParameter name="fill-opacity">1.00</CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>

```

```

</Fill>
<Stroke>
    <CssParameter name="stroke">#000000
    </CssParameter>
    <CssParameter name="stroke-width">1.00
    </CssParameter>
</Stroke>
</PolygonSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

Nótese que cada capa se describe mediante la etiqueta *NamedLayer* y las denominadas *Rule* determinan las clases y estilos de la misma. El empleo de este tipo de recursos es bastante ventajoso, sin embargo como limitante presenta que se deben especificar los nombres de las capas, para que puedan ser relacionadas con las definidas en el *mapObj*. También es cierto que no soporta aun algunas propiedades, no obstante la comunidad de desarrollo de Mapserver se encuentra trabando en este sentido.

4.4 Servicios de mapas

Internet ofrece una gran posibilidad de transmitir y recibir todo tipo de información, con lo cual la integración de SIG otorga a la red todas las posibilidades de trabajar con información espacial, logrando con esto beneficios que son posibles de obtener ocupando nuevas tecnología a través de un medio interactivo como este.

4.4.1 WMS

El desarrollo de la tecnología WMS¹⁹ ha sido la clave para el uso de información espacial en la red. A partir de esta tecnología, se han desarrollado varios sistemas dedicado a poner cartográfica sobre Internet, tales como *MapObjects IMS*, *Spatial Direct*, *ArcIMS* o *MapGuide*, que nos permiten crear aplicaciones SIG en Internet / Intranet para visualizar, consultar y analizar información geográfica por la red.

Las especificaciones del WMS ofrecen una manera de permitir la superposición visual de información geográfica compleja y distribuida simultáneamente en Internet. Potenciando que sea accesible desde la red y de forma dinámica. Todos los datos poseen una relación espacial, lo que en su efecto hace posible el despliegue de información en tiempo real de cualquier parte del mundo. El usuario podrá generar, analizar y utilizar el contenido cartográfico, además de poder acceder a bases de datos y trabajar en tiempo real en post de alguna solución, logrando una total interoperabilidad con toda la información disponible, gracias a las herramientas Web existentes. Los usuarios pueden combinar datos e información accesible vía Internet con datos locales, visualizarlos, hacer consultas y el análisis pertinente.

Las operaciones WMS pueden ser invocadas usando un navegador estándar realizando peticiones en la forma de URL²⁰. El contenido de tales *URL* depende de la operación solicitada. Concretamente, al solicitar un mapa, la *URL* indica qué información debe ser mostrada en el mapa, qué porción de la tierra debe dibujar, el sistema de coordenadas de referencia, y la anchura y la altura de la imagen de salida. Cuando dos o más mapas se producen con los mismos parámetros geográficos y tamaño de salida, los resultados se

¹⁹ WMS acrónimo de Web Map Server

²⁰ URL acrónimo de Uniform Resource Locators

pueden solapar para producir un mapa compuesto. El uso de formatos de imagen que soportan fondos transparentes (e.g., GIF o PNG) permite que los mapas subyacentes sean visibles. Además, se puede solicitar mapas individuales de diversos servidores.

4.4.1.1 Definiendo un consumidor de servicios

Para consumir un servicio de mapa basado en la especificación *wms*, es necesario definir la propiedad *connectiontype* de la respectiva *layer* con el valor *wms* y en *connection* dejar explícito la dirección url en que se encuentra publicado el servicio.

```
LAYER
    NAME "hillshade"
    TYPE RASTER
    STATUS OFF
    TRANSPARENCY 70
    CONNECTIONTYPE WMS
    CONNECTION "http://gisdata.usgs.net:80/com.esri.wms.Esrimap/USGS_WMS_NED?reaspect=false"
    PROJECTION
        "init=epsg:4326"
    END
    METADATA
        "wms_srs"           "EPSG:4326"
        "wms_title"          "US_NED_Shaded_Relief"
        "wms_name"           "US_NED_Shaded_Relief"
        "wms_server_version" "1.1.1"
        "wms_format"         "image/png"
    END
END
```

Como bien se puede observar en el fragmento anterior, el objeto metadato asociado al la capa seleccionada como cliente de un servicio *wms*, será el encargado de describir las particularidades de la conexión. A continuación se detallan algunas de sus propiedades.

- **wms_abstract:** Define una propaganda de texto que proporciona más información sobre el servidor WMS.
- **wms_accessconstraints:** Define las restricciones de acceso a la información, en caso de no requerir ninguna se le especifica el término *none*.
- **wms_encoding:** Define el tipo de codificación xml, por defecto toma valor *so-8859-1*
- **wms_feature_info_mime_type:** Se utiliza para especificar un tipo *MIME* adicional que se puede utilizar cuando se responde a la petición *GetFeature*. Por ejemplo, si desea utilizar la plantilla HTML de la capa de base para su respuesta, se requiere especificar "*wms_feature_info_mime_type*", "*text/html*". Si no se especifica, por defecto MapServer lo define como (*text/plain*) implementando GML.
- **wms_fees:** Define las tasas de información, se especifica el valor "*none*" si no hay cargos.
- **wms_keywordlist:** Define una lista separada por comas de palabras clave o frases en función de agilizar los procesos de búsqueda en el catálogo.
- **wms_onlineresource:** Define la *URL* que se utilizará para acceder a al servidor WMS, este valor se utiliza en la respuesta *GetCapabilities*, se recomienda su definición.
- **wms_resx, wms_resy:** Se utilizan en la etiqueta *BoundingBox* para proporcionar información acerca de la resolución espacial de los datos, los valores deben estar expresados en las unidades definidas por la proyección del mapa
- **wms_service_onlineresource:**
- **wms_title:** Define el nombre de presentación para la capa a consumir.
- **wms_name:** Define el nombre de la capa wms publicada en el servidor.

- **wms_timeformat:** El formato de hora que se utilizará cuando se envía una solicitud, por ejemplo "wms_timeformat" "% Y- % m- % d % H, % Y-% m-% d% H:% M "
- **wms_srs:** Contiene una lista de los códigos de proyección de tipo EPSG disponibles para todas las capas en el servidor. El valor puede contener uno o más pares definidos por la sintaxis: *EPSG:<code>* y separados por espacios, además deben ser especificados en mayúsculas, ejemplo "*EPSG:4269 EPSG:4326*".
- **wms_dataurl_format:** Formato de los recursos en línea donde los datos correspondientes a la capa se puede encontrar.
- **wms_dimension:** Define dimensión en uso, se incorpora a partir de la versión 4.10.
- **wms_dimensionlist:** Lista de las dimensiones disponibles, se incorpora a partir de la versión 4.10.
- **wms_dataurl_href:** Enlace a un recurso en línea donde los datos correspondientes a la capa se puede encontrar.
- **wms_format:** Define el formato a utilizar.
- **wms_formatlist:** Lista de los formatos disponibles para esta capa.
- **wms_metadataurl_href:** Enlace a un recurso en línea, donde los metadatos descriptivos de la capa correspondiente puede encontrarse.
- **wms_onlineresource:** Define URL necesaria para acceder al servidor.
- **wms_server_version:** La versión de la especificación WMS profista por el servidor.

A continuación se muestra un ejemplo que ilustra el empleo de algunas de las propiedades anteriormente descritas.

```

LAYER
  NAME "prov_bound"
  TYPE RASTER
  STATUS ON
  CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
  CONNECTIONTYPE WMS
METADATA
  "wms_name" "prov_bound"
  "wms_server_version" "1.1.1"
  "wms_format" "image/gif"
  "wms_srs" "EPSG:42304"
  "wms_title" "Canadian boundaries" #REQUIRED
  "wms_onlineresource" "http://www2.dmsolf.ca/cgi-bin/mswms_gmap?" #REQUIRED
  "wms_dimensionlist" "time,width"
  "wms_dimension" "time"
  "wms_dimension_time_unitsymbol" "t"
  "wms_dimension_time_units" "ISO8601"
  "wms_dimension_time_uservalue" "1310"
  "wms_dimension_time_default" "1310"
  "wms_dimension_time_multiplevalues" "1310,1410"
  "wms_dimension_time_nearestvalue" "0"
END
END

```

4.4.1.2 Definiendo un proveedor de servicios

Los servidores WMS interactúan con sus clientes a través del protocolo *HTTP* y en la mayoría de los casos son aplicaciones en modo *CGI*. Esto se aplica también a Mapserver, por tanto las peticiones son manejados por el programa denominado *mapserv*. Un elemento a tener en cuenta es que no en todas las versiones de este último se incluye soporte para WMS, no siendo así cuando se compila con la biblioteca *PROJ*. Una forma de verificar esto es utilizar la opción "-v" de línea de comandos y verifique que incluya la

opción "SUPPORTS=WMS_SERVER".

La especificación WMS define un número limitado de tipos de peticiones, y para cada uno de estos un conjunto de parámetros de consulta y comportamientos asociados. A continuación se describen algunas de las mas utilizadas.

1. **GetCapabilities:** Devuelve un documento XML con los metadatos de la información asociada al servidor de mapas digitales.
2. **GetMap:** Permite obtener una imagen de un mapa de acuerdo con las necesidades del usuario.
3. **GetFeatureInfo:** información de retorno sobre la función (s) en una consulta (clic del ratón) ubicación. MapServer soporta 3 tipos de respuesta a esta solicitud:
 - **text/plain:** Incluye únicamente información de atributos en la salida.
 - **text/html:** Genera una salida tomando como base la consulta realizada sobre las plantillas MapServer, especificadas en el parámetro *template* del objeto CLASS. Por otra parte el tipo *MIME* devuelto por las plantillas de clase por defecto es text/html, el cual puede ser controlado mediante la propiedad *wms_feature_info_mime_type* implícita en el metadato.
 - **application/vnd.ogc.gml:** Enfocado en las características específicas de la especificación del *GML*.
4. **DescribeLayer:** Devuelve la descripción XML de una o más capas del mapa, para el cual se debe tener en cuenta lo siguiente según su tipo:
 - **vectoriales:** Se debe configurar al menos una de las propiedades [*wfs_onlineresource* / *ows_onlineresource*] del objeto metadatos, sea a nivel de mapa o de capa.
 - **raster:** Se debe especificar la propiedad *wcs_onlineresource* del metadato, con la misma lógica que el anterior.
5. **GetLegendGraphic:** Devuelve una imagen de la leyenda para la capa solicitada, con la etiqueta previamente definida.

Independiente de lo expuesto anteriormente y en función de garantizar un servicio con calidad, existe un conjunto de propiedades que no deben ser pasadas por alto. En el caso del objeto mapa, es obligatorio definir el nombre, la proyección y el metadatos, en este último se deben especificar las propiedades: *wms_title*, *wms_onlineresource*, *wms_srs*. Por otra parte cada una de las capas publicadas deben contener: el nombre (*name*), el estado (*status*), el objeto proyección (*projection*), la propiedad *template* en el caso que se requiera invocar *GetFeatureInfo*, la propiedad *dump* con valor *true* en función de realizar una petición en modo *GML* a *GetFeatureInfo*, y en el objeto *metadata* definir los atributos *wms_title* y *wms_srs*.

Nótese que las propiedades *name* y *wms_tile* de las capas deben contener un valor único que las identifique, además debe cumplir con la restricción de nomenclatura que las limita a no comenzar con dígitos ni contener espacios en blanco. Teniendo en cuenta que estos constituyen un punto de referencia para las solicitudes de tipo *GetMap* y *GetFeatureInfo*.

Por otra parte la propiedad *wms_onlineresource* se establece en los metadatos del mapa en función de especificar la URL que se debe utilizar para acceder al propio servidor. Esto es necesario para las peticiones de tipo *GetCapabilities*, en caso de no ser especificado el proveedor tratará de proporcionar uno predeterminado basado en la combinación del nombre definido para el script y el *hostname*, aunque resuelve esta práctica no es

recomendable.

Si el bloque de proyección se presenta en el formato "`init=EPSG:xxxx`" entonces MapServer también utilizará esta información para generar una etiqueta de tipo `<BoundingBox>` para la capa de nivel superior en el documento que describe las capacidades WMS definido por el tipo de solicitud denominada `GetCapabilities`. Tenga en cuenta que el `BoundingBox` es un elemento opcional, pero es considerado una buena práctica tanto sea posible.

Lo descrito anteriormente es suficiente para MapServer, permitiéndole reconocer el código EPSG e incluirlo en etiquetas SRS en la salida de `GetCapabilities`, por tanto no sería requerido definir la propiedad `wms_srs` del metadatos. Sin embargo, a menudo es imposible encontrar un código EPSG para que coincida con la proyección de los datos. En esos casos, el "`wms_srs`" se emplea para listar uno o más códigos EPSG en que los datos pueden ser servidos, delegando al objeto proyección el verdadero valor definido por la PROJ4.

A continuación se describen los parámetros necesarios para la solicitud denominada `GetMap` definida por la especificación WMS:

- **VERSION:** Define la versión de la solicitud.
- **REQUEST:** Define el tipo de solicitud, en este caso debe ser especificado como `GetMap`.
- **LAYERS:** Capa o listado de estas separadas por coma, toma carácter opcional en caso de ser especificado el `SLD`.
- **STYLES:** Estilo o listado de estos separados por coma para cada una de las capas seleccionada, es opcional en caso de especificarse el `SLD`.
- **SRS:** Spatial Reference System. [`namespace:identifier`]
- **BBOX:** Define los puntos extremos del área de visualización geográfica definido como `BoundingBox` definido por la estructura `[minx,miny,maxx,maxy]`
- **WIDTH:** Ancho en píxeles de la imagen del mapa..
- **HEIGHT:** Altura en píxeles de la imagen del mapa.
- **FORMAT:** Formato de salida del mapa.

A continuación se muestra un ejemplo, que describen los elementos que intervienen en una petición para el consumo de un servicio vía CGI: <http://my.host.com/cgi-bin/mapserv?map=mywms.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&LAYERS=prov>

A continuación se describen los parámetros necesarios para la solicitud denominada `GetLegendGraphic` definida por la especificación WMS:

- **LAYER:** Nombre de la capa WMS para devolver la imagen de leyenda. Tenga en cuenta que este es el parámetro `<Nombre>` de la capa en el `GetCapabilities`.
- **FORMAT:** Formato de la imagen para la leyenda (por ejemplo, "`image/png`").
- **WIDTH:** Ancho de la imagen de la leyenda. Observe que el parámetro Ancho sólo se utiliza cuando el parámetro de regla se utiliza también en la solicitud, es de carácter opcional.
- **HEIGHT - (Optional)** Height of the legend image. Note that the Height parameter is only used when the Rule parameter is also used in the request.
- **SLD:** Define la dirección URL del SLD, permitiendo aplicar los estilos para luego generar la leyenda correspondiente, este atributo es de carácter opcional.
- **SLD_BODY:** En lugar de especificar una URL se describe el cuerpo del SLD, este

- **SCALE:** Permite especificar una escala para que únicamente las capas que caen en esta escala tendrá una leyenda, este atributo es de carácter opcional.
- **RULE:** Especifique el nombre de la clase para generar la imagen de leyenda (en lugar de generar un ícono y una etiqueta para TODAS las clases de la capa), este atributo es de carácter opcional.

A continuación se muestra un ejemplo que describe como se realizan este tipo de solicitudes: <http://gis.os/cgi-bin/mapserv?SERVICE=WMS&VERSION=1.1.1&layer=park&FORMAT=image/png&REQUEST=getlegendgraphic>

Por otra parte WMS especifica que el formato básico que se utiliza para las solicitudes de tiempo se basa en la norma ISO 8601:1988 (E) "extendida". MapServer soporta un conjunto limitado de patrones que se definen en las especificaciones ISO 8601, así como algunos otros patrones que son útiles pero no cumplen con la norma ISO 8601. Aquí está una lista de patrones soportados actualmente:

Time Patterns	Examples
YYYYMMDD	20041012
YYYY-MM-DDTHH:MM:SSZ	2004-10-12T13:55:20Z
YYYY-MM-DDTHH:MM:SS	2004-10-12T13:55:20
YYYY-MM-DD HH:MM:SS	2004-10-12 13:55:20
YYYY-MM-DDTHH:MM	2004-10-12T13:55
YYYY-MM-DD HH:MM	2004-10-12 13:55
YYYY-MM-DDTHH	2004-10-12T13
YYYY-MM-DD HH	2004-10-12 13
YYYY-MM-DD	2004-10-12
YYYY-MM	2004-10
YYYY	2004
THH:MM:SSZ	T13:55:20Z
THH:MM:SS	T13:55:20

Ilustración 17 Patrones de tiempo soportados

Para mayor comprensión de este tema véase el ejemplo que se muestra a continuación, el cual ilustra como poner en práctica estas opciones.

```

LAYER
  NAME "earthquakes"
  METADATA
    "wms_title" "Earthquakes"
    "wms_timeextent" "2004-01-01/2004-02-01"
    "wms_timeitem" "TIME"
    "wms_timedefault" "2004-01-01 14:10:00"
  END
  TYPE POINT
  STATUS ON
  DATA "quakes"
  CLASS
    ..
  END
END

```

A continuación se muestra la salida que genera una solicitud de tipo *GetCapabilities* basada en la especificación definida para una etiqueta *layer* similar a la anterior.

```

<Layer queryable="0" opaque="0" cascaded="0">
  <Name>earthquakes</Name>
  <Title>Earthquakes</Title>
  <SRS>EPSG:4326</SRS>
  <LatLonBoundingBox minx="-131.02" miny="24.84" maxx="-66.59" maxy="48.39" />

```

```

<BoundingBox SRS="EPSG:4326" minx="-131.02" miny="24.84" maxx="-66.59" maxy="48.39" />
<Dimension name="time" units="ISO8601"/>
<Extent name="time" default="2004-01-01T14:10:00" nearestValue="0">2004-01-01/2004-02-01</Extent>
</Layer>

```

4.4.1.3 Map context

El término *map context* o contexto de mapas, proviene de una especificación establecida por la OGC denominada *Web Map Context*, coincidiendo con la WMS en varios aspectos. Un contexto de mapa es un documento XML que describe la apariencia de capas de uno o más servidores WMS, pudiendo ser transferido entre los clientes en función compartir datos asociados a las diferentes vistas, tales como su estado, capas, u otra información de carácter adicional. Esto permite a las aplicaciones cliente cargar y almacenar configuraciones de mapas en un formato estándar como el XML.

```

MAP
  NAME WMS_CONTEXT
  STATUS ON
  SIZE 400 300
  SYMBOLSET ..../etc/symbols.sym
  EXTENT -2200000 -712631 3072800 3840000
  UNITS METERS
  SHAPEPATH "..//data"
  IMAGECOLOR 255 255 255
  FONTSET ..//etc/fonts.txt

WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
  METADATA
    "wms_abstract" "Demo for map context document. Blah blah..."
    "wms_title" "Map Context demo" ##### REQUIRED
  END
END

PROJECTION
  "init=epsg:42304"
END

LAYER
  NAME "prov_bound"
  TYPE RASTER
  STATUS ON
  CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
  CONNECTIONTYPE WMS
  METADATA
    "wms_name" "prov_bound"
    "wms_server_version" "1.1.1"
    "wms_format" "image/gif"
    "wms_srs" "EPSG:42304"
    "wms_title" "Canadian boundaries" ##### REQUIRED
    "wms_onlineresource" "http://www2.ca/cgi-bin/msw_gmap?" ##### REQUIRED
    "wms_dimensionlist" "time,width"
    "wms_dimension" "time"
    "wms_dimension_time_unitsymbol" "t"
    "wms_dimension_time_units" "ISO8601"
    "wms_dimension_time_uservalue" "1310"
    "wms_dimension_time_default" "1310"
    "wms_dimension_time_multiplevalues" "1310,1410"
    "wms_dimension_time_nearestvalue" "0"
  END
END

LAYER
  NAME popplace

```

```

TYPE RASTER
STATUS ON
CONNECTIONTYPE WMS
CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
METADATA
    "wms_name" "popplace"
    "wms_server_version" "1.1.1"
    "wms_format" "image/png"
    "wms_srs" "EPSG:42304"
    "wms_title" "Canadian Cities" ##### REQUIRED
    "wms_onlineresource"      "http://www2.dmsolutions.ca/cgi-bin/msgmap?"      #####
    REQUIRED
END
END # Layer

END # Map File

```

A continuación se muestra el contenido del fichero *mscontexpro.php*, el cual se encarga de leer un *mapfile* y generar un fichero que encapsule el contexto de mapa en función del mismo.

```

<?php
    dl("php_mapscript.dll");
    $objMap = ms_newMapObj("gmap_wms_context.map");
    $objMap ->saveMapContext("gmap_wms_context_output.xml");
?>

```

Obsérvese que para lograr el objetivo planteado, tan solo fue requerida la invocación de la función *saveMapContext* como recurso nativo del *mapObj*. A continuación se muestra el contenido del fichero generado denominado *gmap_wms_context_output.xml*.

```

<?xml version='1.0' encoding="ISO-8859-1" standalone="no" ?>
<ViewContext version="1.1.0" id="WMS_CONTEXT" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <General>
        <Window width="400" height="300"/>
        <BoundingBox SRS="EPSG:42304" minx="-22.00" miny="-711.0" maxx="300.0" maxz="170.00"/>
        <Title>Map Context demo</Title>
        <Abstract>Demo for map context document. </Abstract>
        <ContactInformation>      </ContactInformation>
    </General>

    <LayerList>
        <Layer queryable="0" hidden="0">
            <Server service="OGC:WMS" version="1.1.1" title="Canadian boundaries">
                <OnlineResource xlink:type="simple" xlink:href="http://www2.dtots.ca/cgi-bin/msw /">
            </Server>

            <Name>prov_bound</Name>
            <Title>Canadian boundaries</Title>
            <SRS>EPSG:42304</SRS>
            <FormatList>
                <Format current="1">image/gif</Format>
            </FormatList>
            <DimensionList>
                <Dimension name="time" units="ISO8601" unitSymbol="t"
                    userValue="1310" default="1310" multipleValues="26 " />
            </DimensionList>
        </Layer>

        <Layer queryable="0" hidden="0">
            <Server service="OGC:WMS" version="1.1.1" title="Canadian Cities">
                <OnlineResource xlink:type="simple" xlink:href="http://www.ca/cgi-bin/msws /">
            </Server>
            <Name>popplace</Name>
            <Title>Canadian Cities</Title>
            <SRS>EPSG:42304</SRS>
            <FormatList>

```

```

<Format current="1">image/png</Format>
</FormatList>
</DimensionList>
</Layer>

</LayerList>
</ViewContext>

```

MapServer CGI también permite cargar un mapa de contexto mediante el uso de un parámetro explícito en la petición denominado **CONTEXT**, permitiendo acceder al mismo desde un directorio local. A continuación se muestra un ejemplo de petición:

http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=/path/to/contextfile.xml&LAYERS=layer_name1 layers_name2

Aunque también se puede consultar a través de una *url*: http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=http://URL/path/to/contextfile.xml&LAYERS=layer_name1 layer_name2

Un elemento importante a destacar en este sentido, es que debido a las políticas de seguridad dirigidas a proteger los clientes, se desactiva por defecto la opción que permite cargar un archivo de este tipo desde una URL.

```

MAP
  NAME "map-context"
  STATUS ON
  SIZE 400 300
  EXTENT -2200000 -712631 3072800 3840000
  UNITS METERS
  IMAGECOLOR 255 255 255
  IMAGETYPE png
  CONFIG "CGI_CONTEXT_URL" "1"

WEB
  ...
END
LAYER
  ...
END
END

```

Por tanto de ser necesario debe habilitarse de forma explícita esta funcionalidad. Para esto el usuario tiene que establecer un parámetro de configuración del mapa denominado **CONFIG** especificándole el valor **1** en la propiedad **CGI_CONTEXT_URL**, para mayor comprensión véase el fragmento de código anterior.

4.4.2 WFS

Los servicios WFS²¹ han sido definidos por el OGC²² y se consideran un estándar, que ofrece una interfaz de comunicación que permite interactuar con los mapas servidos por el estándar WMS, como por ejemplo, editar la imagen que nos ofrece el servicio WMS o analizar la imagen siguiendo criterios geográficos. Para realizar estas operaciones se utiliza el lenguaje GML que deriva del XML, que es el estándar a través del que se transmiten las órdenes WFS.

WFS no transaccional permite hacer consultas y recuperación de elementos geográficos. Por el contrario WFS-T²³ permite además la creación, eliminación y actualización de estos

²¹ WFS acrónimo de Web Feature Service

²² OGC acrónimo de Open Geospatial Consortium

²³ WFS-T acrónimo de Web Feature Service Transactional

elementos geográficos del mapa.

4.4.1 Definiendo un proveedor de servicio

De igual forma que en el apartado anterior, los servicios de tipo WFS son definidos a partir de la especificación explicita en el objeto *metadato* de parámetros que describan dicha comunicación.

- **gml_exclude_items:** Define una lista delimitada por comas de los elementos a excluir. A partir de MapServer 4.6, puede controlar cuantos atributos son expuestos en los datos metadatos de la capa. A continuacion se muestra un ejemplo que ilustra tal afirmacion:

```
"gml_include_items" "all"  
"gml_exclude_items" "Phonenum"
```

- **gml_featureid:** Campo que se utilizará como el identificador de las geometrías en la salida GML. Puede ser especificado como *wfs_featureid* o *ws_feature_id*.
- **gml_groups:** Define una lista separada por comas de nombres definidos para los grupos de las capas.
- **gml_[group name]_group:** Define una lista separada por comas de los atributos del grupo. He aquí un ejemplo:

```
"gml_include_items" "all"  
"gml_groups" "display"  
"gml_display_group" "Name_e,Name_f"
```

- **gml_include_items:** Define una lista delimitada por comas de los elementos a incluir. A partir de MapServer 4.6, puede controlar cuantos atributos son expuestos en los datos metadatos de la capa. En caso que se deseen incluir todos se podría emplear la palabra reservada *all*. A continuacion se muestra un ejemplo que ilustra tal afirmacion:

```
"gml_include_items" "all"  
"gml_include_items" "Name,ID"
```

- **gml_[item name]_alias:** Define un alias para el nombre de un atributo, teniendo en cuenta que en GML servido se hace referencia a este atributo por el alias previamente especificado. He aquí un ejemplo:

```
"gml_province_alias" "prov"
```

- **gml_[item name]_type:** Permite especificar el tipo del atributo, los valores válidos son [*Integer*/*Real*/*Character*/*Date*/*Boolean*]
- **gml_xml_items:** Define una lista delimitada por comas de elementos que no deben ser codificados en XML
- **gml_geometries:** Permite proporcionar un nombre distinto del predeterminado "msGeometry" para las geometrías. El valor se especifica como una cadena que se utiliza para los nombres de elementos geométricos
- **gml_[name]_type:** Cuando se emplea *gml_geometries*, también es necesario especificar el tipo de geometría de la capa. Esto se logra al proporcionar un valor para *gml_[nombre]_type*, donde *[nombre]* es el valor de cadena especificado para *gml_geometries*, y un valor que es uno de los comprendidos en [*point* / *multipoint* / *line* / *multiline* / *polygon* / *multipolygon*]
- **gml_[name]_occurrences:** MapServer aplica los valores por defecto de 0 y 1, respectivamente, a los atributos "*minOccurs*" y "*maxOccurs*" asociados a los elementos geométricos, como se puede ver en los ejemplos anteriores. Para anular esta configuración predeterminada, se asigna un valor a la propiedad

gml_[name]_occurrences de metadatos, donde de nuevo *[nombre]* es el valor de la cadena especificada para gml_geometries, y el valor es una pareja separada por comas que contiene los límites respectivos inferior y superior.

- **wfs_abstract:** Su comportamiento es igual que el de la propiedad *wfs_abstract* en el objeto Web.
- **wfs_extent:** Se utiliza para etiqueta de la capa que define el BoundingBox principalmente para aquellos casos en los que es imposible o muy ineficiente para MapServer investigar el origen de datos en función de determinar su extensión territorial. El valor de estos metadatos es "*minx miny maxx maxy*" separadas por espacios, con los valores en las unidades de proyección de la capa. Si *wfs_extent* se proporciona entonces tiene prioridad y MapServer no intentará leer extensión del archivo de origen.
- **wfs_metadataurl_format:** Define el formato de archivo que describe los registros del metadatos. Los valores válidos son "XML", "SGML", o "HTML". La capa de metadatos *wfs_metadataurl_type* y *wfs_metadataurl_href* también debe ser especificado.
- **wfs_metadataurl_href:** Define la dirección URL a los metadatos de la capa. La capa de metadatos *wfs_metadataurl_type* y *wfs_metadataurl_format* también debe ser especificado.
- **wfs_metadataurl_type:** Define la norma que cumple los metadatos especificados. Actualmente, sólo dos tipos son válidos: "TC211", que se refiere a [ISO 19115], y "FGDC", que se refiere a [FGDC CSDGM]. Las propiedades de metadatos *wfs_metadataurl_format* y *wfs_metadataurl_href* también debe ser especificadas.
- **wfs_srs:** Si no hay valor SRS definido en el nivel superior del mapfile, entonces este será utilizado para publicitar este tipo de característica.
- **wfs_title:** Se comporta de igual forma que la propiedad *wfs_title* en el objeto Web.

A continuación se muestra un ejemplo que ilustra un proveedor de servicios con dichas características, en el cual se resaltan mediante comentarios la forma en algunas propiedades son requeridas, sean de carácter obligatoria, recomendada u opcional.

```

NAME "WFS_server"

STATUS ON
SIZE 400 300
SYMBOLSET ..../etc/symbols.s y m
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET ..../etc/fonts.txt

WEB
    IMAGEPATH "/ms4w/tmp/ms_tmp/"
    IMAGEURL "/ms_tmp/"

METADATA
    "wfs_title" "GMap WFS Demo Server" ## REQUIRED
    "wfs_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?" ## Recommended
    "wfs_srs" "EPSG:42304 EPSG:42101 EPSG:4269 EPSG:4326" ## Recommended
    "ows_schemas_location" "http://ogc.dmsolutions.ca" ## Optional
END

PROJECTION
    "init=epsg:42304"
END

LAYER

```

```

NAME "province"
METADATA
    "wfs_title" "Provinces" ## REQUIRED
    "gml_featureid" "ID" ## REQUIRED
    "gml_include_items" "all" ## Optional (serves all attributes for layer)
END
TYPE POLYGON
STATUS ON
DATA province
PROJECTION
    "init=epsg:42304"
END
DUMP TRUE ## REQUIRED
CLASS
    NAME "Canada"
    STYLE
        COLOR 200 255 0
        OUTLINECOLOR 120 120 120
    END
    TEMPLATE "ttt_query.html"
END
END # Layer
END # Map File

```

A continuación se muestra el contenido devuelto como resultado de la invocación de una solicitud de tipo GetCapabilities.

```

<?xml version='1.0' encoding="ISO-8859-1" ?>
<WFS_Capabilities version="1.0.0" updateSequence="0" xmlns="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs http://ogc.dmsolutions.ca/wfs/1.0.0/WFS-capabilities.9">

<!-- MapServer version 4.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF
OUTPUT=SWF SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE -->

<Service>
    <Name>MapServer WFS</Name>
    <Title>GMap WMS Demo Server</Title>
    <OnlineResource>http://localhost/cgi-bin/mapserv.exe?map=/msapps/wfs_filter/htdocs/ns_wfsserver.map & </OnlineResource>
</Service>

<Capability>
    <Request>
        <GetCapabilities>
            <DCPType>
                <HTTP>
                    <Get onlineResource="http://localhost/cgi-bin/mapserv.exe?
map=/msapps/wfs_filter/htdocs/ns_wfsserver.map&" />
                </HTTP>
            </DCPType>
            <DCPType>
                <HTTP>
                    <Post onlineResource="http://localhost/cgi-bin/mapserv?
map=/msapps/wfsfilter/htdocs/wfsserver.map&" />
                </HTTP>
            </DCPType>
        </GetCapabilities>
    </Request>
</Capability>

<Filter_Capabilities>
    <Spatial_Capabilities>
        <Spatial_Operators>
            <Intersect/>

```

```

        <DWithin/>
        <BBOX/>
    </Spatial_Operators>
</Spatial_Capabilities>
<Scalar_Capabilities>
    <Logical_Operators />
    <Comparison_Operators>
        <Simple_Comparisons />
        <Like />
        <Between />
    </Comparison_Operators>
</Scalar_Capabilities>
</Filter_Capabilities>
</WFS_Capabilities>

```

4.4.1 Definiendo un consumidor de servicio

De igual forma que en los apartados anteriores para realizar un consumidor de este tipo de servicio internamente en el mapfile, se debe definir la propiedad *connectiontype* con el valor *wfs* y especificar en *connection* la *url* en que se encuentra publicado el mismo. El resto de las propiedades parametrizables se especifican en el la etiqueta *web*. Para mayor comprensión véase el ejemplo que se muestra a continuación.

```

LAYER
    NAME "park"
    TYPE POLYGON
    STATUS ON
    CONNECTIONTYPE WFS
    CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?"
    METADATA
        "wfs_typename" "park"
        "wfs_version" "1.0.0"
        "wfs_request_method" "GET"
        "wfs_connectiontimeout" "60"
        "wfs_maxfeatures" "1"
    END
    PROJECTION
        "init=epsg:42304"
    END
    CLASS
        NAME "Parks"
        STYLE
            COLOR 200 255 0
            OUTLINECOLOR 120 120 120
        END
    END
END # Layer

```

Obsérvese como en este ejemplo se deja explícito el método de envío para la transacción a través de la propiedad *wfs_request_method*, así como *wfs_connectiontimeout* para establecer de forma explícita el tiempo máximo en que será sostenida la conexión con el servidor remoto, en caso de no ser especificada toma por defecto el valor de 30 segundos o el valor que se le especifique a nivel de objeto mapa.

Un elemento importante a tener en cuenta es que los archivos temporales generados por cada solicitud en formato GML, se escriben en el directorio definido por la propiedad denominada *imagepath*, esto podría convertirse en un problema de seguridad. En función de esto se debe valorar las opciones *wfs_tmpdir* y *wfs_cache_dir* que permiten dejar de forma explícita donde podrían ser almacenados de forma que sea controlado su acceso.

4.4.3 WCS

La interfaz estándar WCS²⁴, también conocida como el servicio de cobertura web definido por la OGC, proporciona una interfaz que permite realizar peticiones de cobertura geográfica a través de la web utilizando llamadas independientes de la plataforma. Las coberturas son objetos en un área geográfica mientras que la interfaz WMS o portales de mapas online como Google Maps devuelven sólo una imagen, que los usuarios no pueden editar o analizar espacialmente.

El servicio de cobertura web básico permite realizar consultas y obtener coberturas. WCS describe las operaciones de descubrimiento, petición o transformación de datos. El cliente genera una solicitud y la envía al servidor utilizando el protocolo *HTTP*. Este a su vez ejecuta dicha petición, en este caso especificación en cuestión utiliza el *HTTP* como plataforma de computación distribuida, aunque no es un requisito obligatorio.

En función de garantizar el soporte para operaciones definidas en la especificación WCS, se definen dos tipos de codificación. La primera se basa en el lenguaje XML, apto para plataformas que se sustenten sobre *HTTP POST/SOAP* y el segundo consiste en pares Clave-Valor, principalmente para *HTTP GET/Remote Procedure Call*. Es importante destacar que en la clasificación de servicios web, WCS está categorizada como un servicio de tipo RPC no-tranquilo.

Por otra parte los datos pueden estar disponibles en varios formatos, como *DTED*, *GeoTIFF*, *HDF-EOS*, o *NITFS*. Soportando varios tipos de capas de información, tales como las series de puntos como ejemplos de localizaciones, la red regular de píxeles o puntos representando una foto, el conjunto de curvas segmentadas a menudo utilizadas para rutas de carretera, el conjunto de polígonos de *Thiessen* utilizados para analizar los datos distribuidos espacialmente como medidas de precipitación y la red irregular triangulada (*TIN*) a menudo utilizada para modelos de terreno. Además que se pueden añadir rangos de información a las localizaciones, como la velocidad media del viento o rendimientos del cultivo.

4.4.3 Definiendo un proveedor de servicios

Este tipo de servicio no difiere mucho a los vistos anteriormente, en este caso se deben especificar en el metadatos asociado a la etiqueta web del mapa, los elementos que describen la conexión. A continuación se muestra un ejemplo que ilustra dicha afirmación.

```
MAP
  NAME WCS_server
  STATUS ON
  SIZE 400 300
  SYMBOLSET ..\etc\symbols.sym
  EXTENT -2200000 -712631 3072800 3840000
  UNITS METERS
  SHAPEPATH "..\data"
  IMAGECOLOR 255 255 255
  FONTSET "..\etc\fonts.txt"
WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
METADATA
  "wcs_label" "GMap WCS Demo Server" ### required
  "wcs_description" "Some text description of the service"
  "wcs_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?" ### recommended
  "wcs_fees" "none"
  "wcs_accessconstraints" "none"
```

²⁴ WCS acrónimo de Web Coverage Service

```

    "wcs_keywordlist" "wcs,test"
    "wcs_metadatalink_type" "TC211"
    "wcs_metadatalink_format" "text/plain"
    "wcs_metadatalink_href" "http://someurl.com"
    "wcs_address" "124 Gilmour Street"
    "wcs_city" "Ottawa"
    "wcs_stateorprovince" "ON"
    "wcs_postcode" "90210"
    "wcs_country" "Canada"
    "wcs_contactelectronicmailaddress" "blah@blah"
    "wcs_contactperson" "me"
    "wcs_contactorganization" "unemployed"
    "wcs_contactposition" "manager"
    "wcs_contactvoicetelephone" "613-555-1234"
    "wcs_contactfacimiletelephone" "613-555-1235"
    "wcs_service_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?"

    END
END

PROJECTION
  "init=epsg:42304"
END

LAYER
  NAME bathymetry
  METADATA
    "wcs_label" "Elevation/Bathymetry" ### required
    "wcs_rangeset_name" "Range 1" ### required to support DescribeCoverage request
    "wcs_rangeset_label" "My Label" ### required to support DescribeCoverage request
  END
  TYPE RASTER ### required
  STATUS ON
  DATA bath_mapserver.tif
  PROJECTION
    "init=epsg:42304"
  END
  DUMP TRUE ### required
END
END # Map File

```

En caso que se requiera conocer el contrato que describe los elementos que componen este tipo de servicio, se procede a ejecutar la invocacion de una url similar a la que se muestra a continuación:

```
http://my.host.com/cgi-bin/mapserv?map=mywcs.map&SERVICE=WCS&VERSION=1.0.0&REQUEST=GetCapabilities
```

Por otra parte se puede realizar una solicitud en función de tipo *GetCoverage* especificándole los parámetros necesarios para obtener la información deseada de la siguiente forma:

```
http://my.host.com/cgi-bin/mapserv?map=mywcs.map&SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage&coverage=bathymetry&CRS=EPSG:42304&BBOX=-2200000,-712631,3072800,3840000 &WIDTH=3199&HEIGHT=2833&FORMAT=GTiff
```

Como desventaja principalmente en función de las tecnologías definidas para la plataforma GeneSIG, se encuentra que el servidor de mapas Mapserver aun no soporta a plenitud este tipo de servicio, siendo una de las líneas de desarrollo de su comunidad.

4.5 Catálogo de funciones

En este apartado se estarán describiendo algunos de los recursos mas utilizados que conforman el catalogo de funciones provista por el *API* de la *Mapscript* en función del

lenguaje *PHP*.

- **ms_newMapObj(mapfilepath:String):MapObj** Devuelve un objeto de tipo mapa, recibe por parámetros la dirección del mapfile, en caso de que el “mapfilepath” se vacío entonces lo interpreta como un mapfile dinámico.
- **ms_newLayerObj(_parent:MapObj):LayerObj** Devuelve un objeto de tipo capa, por parámetro se le especifica a que objeto mapa al que va asociado.
- **ms_newClassObj(_parent:LayerObj):ClassObj** Devuelve un objeto de tipo clase, por parámetro se le especifica a que objeto capa al que va asociado.
- **ms_newStyleObj(_parent:ClassObj):StyleObj** Devuelve un objeto de tipo estilo, por parámetro se le especifica a que objeto clase al que va asociado.
- **ms_newgridobj(_parent:LayerObj):GridObj** Devuelve un objeto de tipo grid, por parámetro se le especifica a que objeto capa al que va asociado.
- **ms_GetErrorObj():ErrorObj** Devuelve un listado de objetos error.
- **ms_ResetErrorList()** Libera la lista de errores.
- **ms_newPointObj():PointObj** Devuelve un objeto de tipo Punto.
- **ms_newRectObj():RectObj** Devuelve un objeto de tipo Recta.

5. GeneSIG

GeneSIG es un sistema de software con una arquitectura basada en. Estos son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

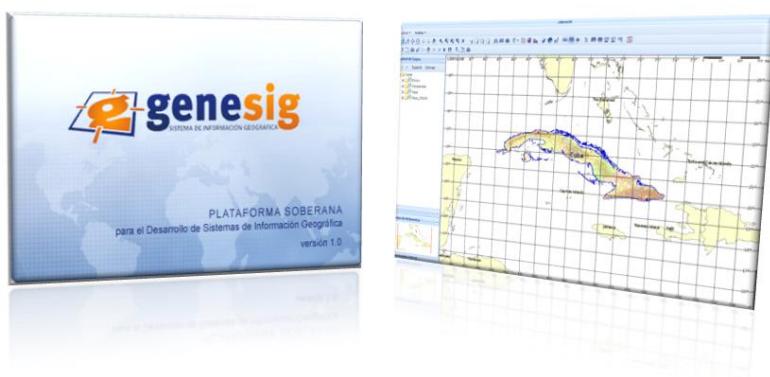


Ilustración 18 Plataforma GeneSIG

Para que una arquitectura de componentes pueda operar es necesario disponer de un entorno normalizado que proporcione soporte a los mecanismos con que se comunican las interfaces.

- Reusabilidad de Servicios: Reducción considerable de tiempos y costos de desarrollo de aplicaciones al utilizar servicios disponibles ya desarrollados, para resolver problemáticas comunes a otras aplicaciones. Aumentado por esta razón la robustez del nuevo sistema, al utilizarse software ya probado.
- Interoperabilidad de aplicaciones: Disminución de la complejidad en el proceso de integración, pues se interactúa con elementos que se abstraen de la tecnología y

ubicación de los servicios.

Entonces se puede decir que las características principales son la modularidad, la reusabilidad y compatibilidad, en la tecnología basada en componentes también se requiere robustez ya que los componentes han de operar en entornos mucho más heterogéneos y diversos. Su premisa es que los componentes cumplan con alta cohesión y bajo acoplamiento.

Uno de los elementos más importantes a resaltar de la arquitectura propuesta por CartoWeb es que, se basa en un modelo de redefinición física a nivel de proyecto, para cada uno de los recursos provistos por dicho *framework*. Esto permite que en caso que sea requerida una modificación, sería posible modificar solamente el fichero que contenga dicha implementación y los demás serían reutilizados del propio marco de trabajo.

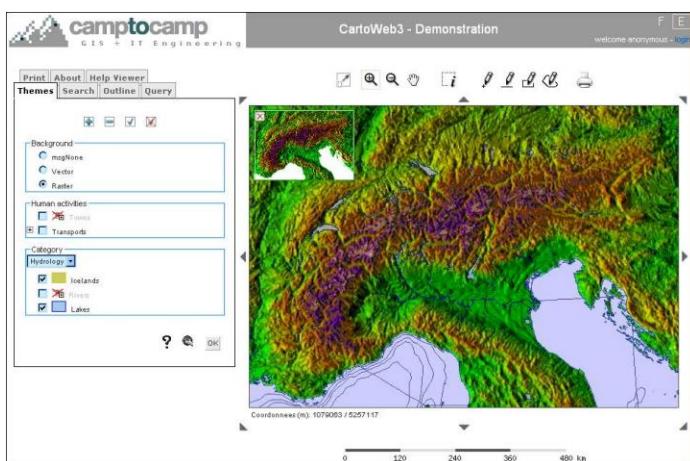


Ilustración 19 Aplicacion demostrativa de CartoWeb

Una característica muy peculiar de esta plataforma es que el fuente del servidor se divide en dos subsistemas, uno encargado de la interacción con los datos provenientes del navegador el cual fue denominado Client y otro que sería responsable de administrar la información correspondiente al tratamiento espacial, así su manipulación a través de la Mapscript, este último fue denominado Server.

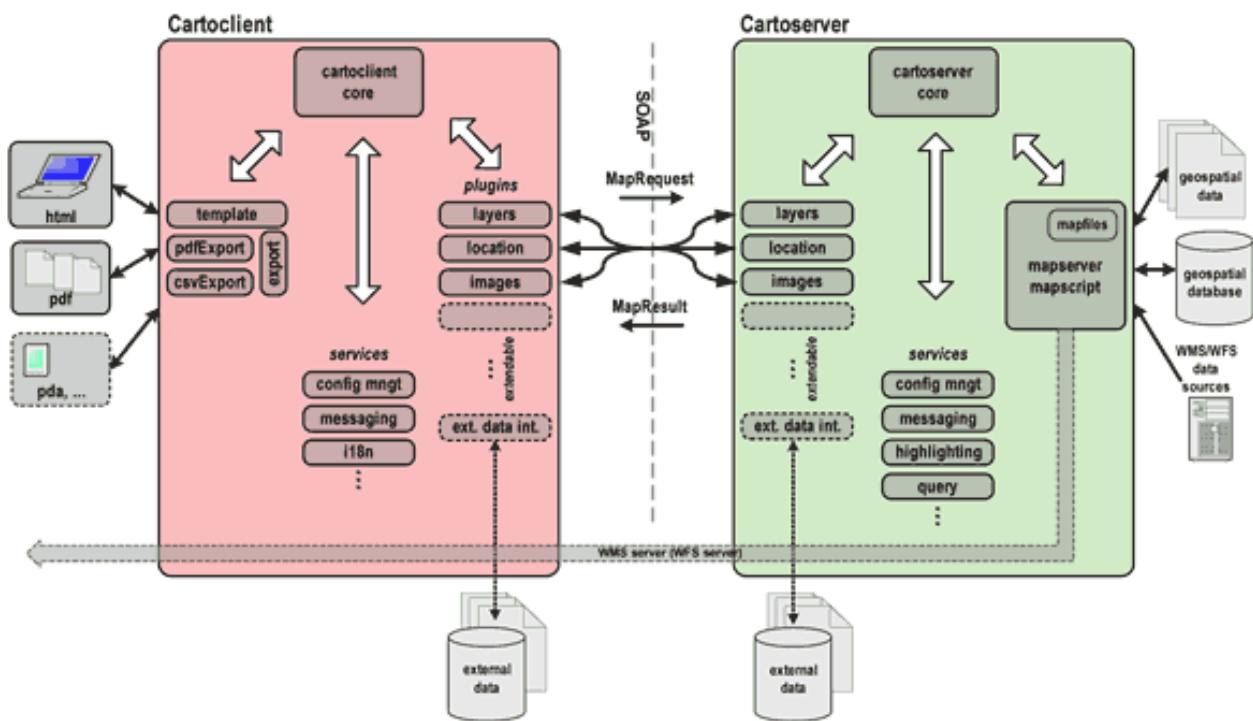


Ilustración 20 Arquitectura definida por CartoWeb.

Por las particularidades de este tipo de sistemas y en aras de garantizar la interoperabilidad con la mayor gama de aplicaciones posibles basadas en esta misma arquitectura, se definen cuatro modos de comunicación entre cliente y servidor las cuales se describen a continuación.

El modo **Direct** establece que la comunicación debe ser directa a través de la propia instanciación de las clases CartoClient y CartoServer. En este modo se gana por concepto de rendimiento.

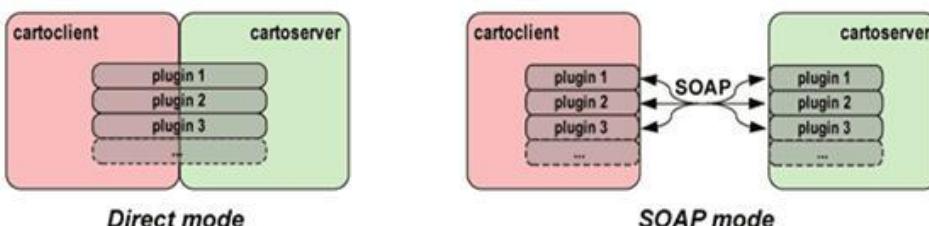


Ilustración 21 Modos de comunicación Directo y SOAP

Por otra parte el modo **SOAP** como su nombre indica, plantea que dicha comunicación debe ser a través del protocolo SOAP, en este caso la comunicación entre CartoClient y el CartoServer será mediante servicios web.

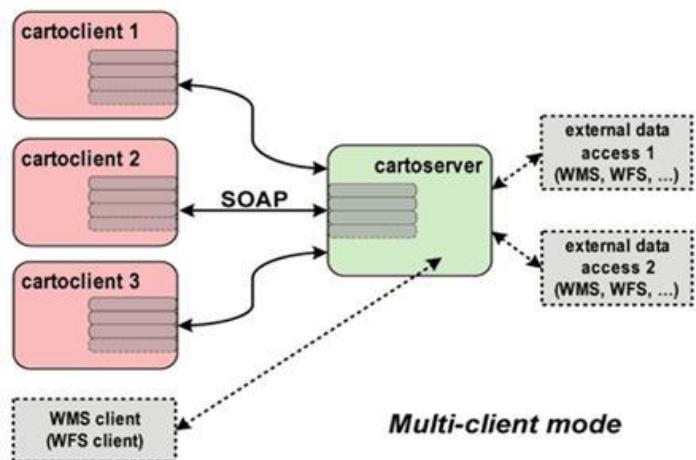


Ilustración 22 Modos de comunicación Multiclient

El modo Multiclient plantea un nivel de reutilización mayor describiéndose como varios *plugins* podrían compartir un mismo módulo servidor. En este caso se incluyen además los clientes de servicios WMS²⁵ y los WFS²⁶ que también se comunican con el CartoServer.

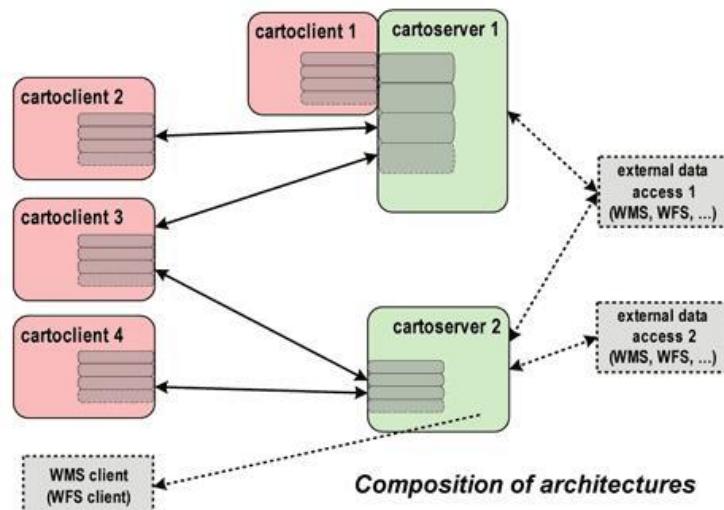


Ilustración 23 Modos de comunicación Composition of architectures

La última opción y no por ello la menos importante es la combinación de los modos anteriores, este brinda la posibilidad de integrar los diferentes tipos de propuestas, esto se conoce como modo Composición de Arquitectura, la misma comunicación modo Directo, SOAP y Multicliente, se observan también la presencia de los WMS y los WFS. En este modo aparece una nueva filosofía en cuanto a comunicación cliente y servidor, se muestra un mismo CartoClient accediendo a varios CartoServer, este tipo de interacción hace que la comunicación sea flexible y adaptable a cualquier medio.

²⁵ WMS acrónimo de Web Map Service

²⁶ WFS acrónimo de Web Feature Service

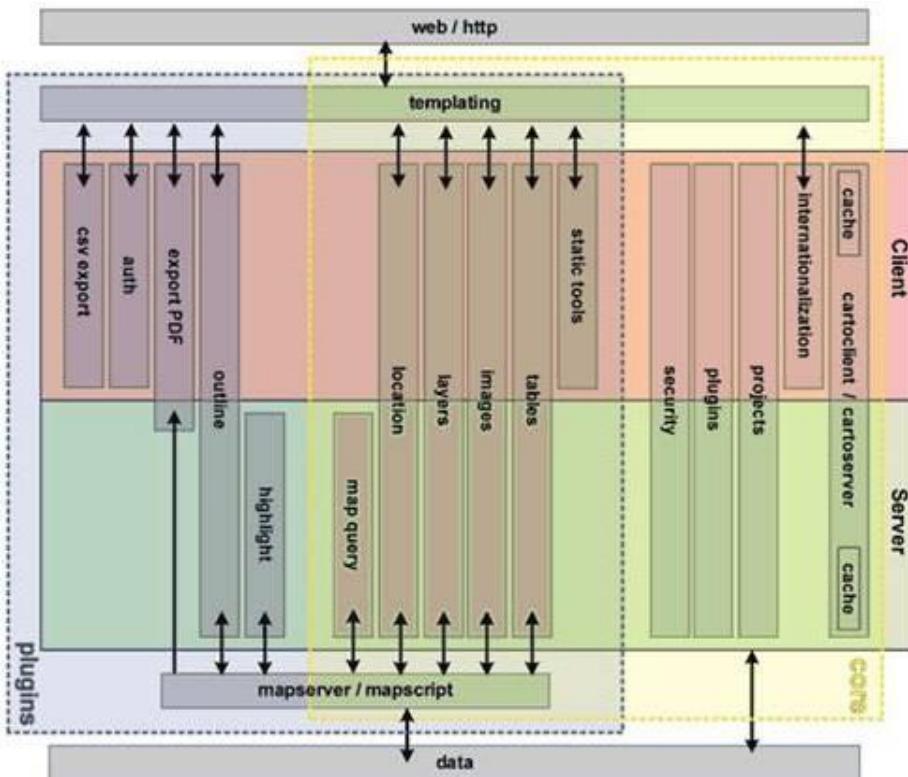


Ilustración 24 Elementos que intervienen en la arquitectura de CartoWeb

Por otra parte los entornos de despliegue para este tipo de productos son un tanto atípicos al desarrollo tradicional de aplicaciones web, a continuación se muestra una imagen que ilustra mejor dicha afirmación.

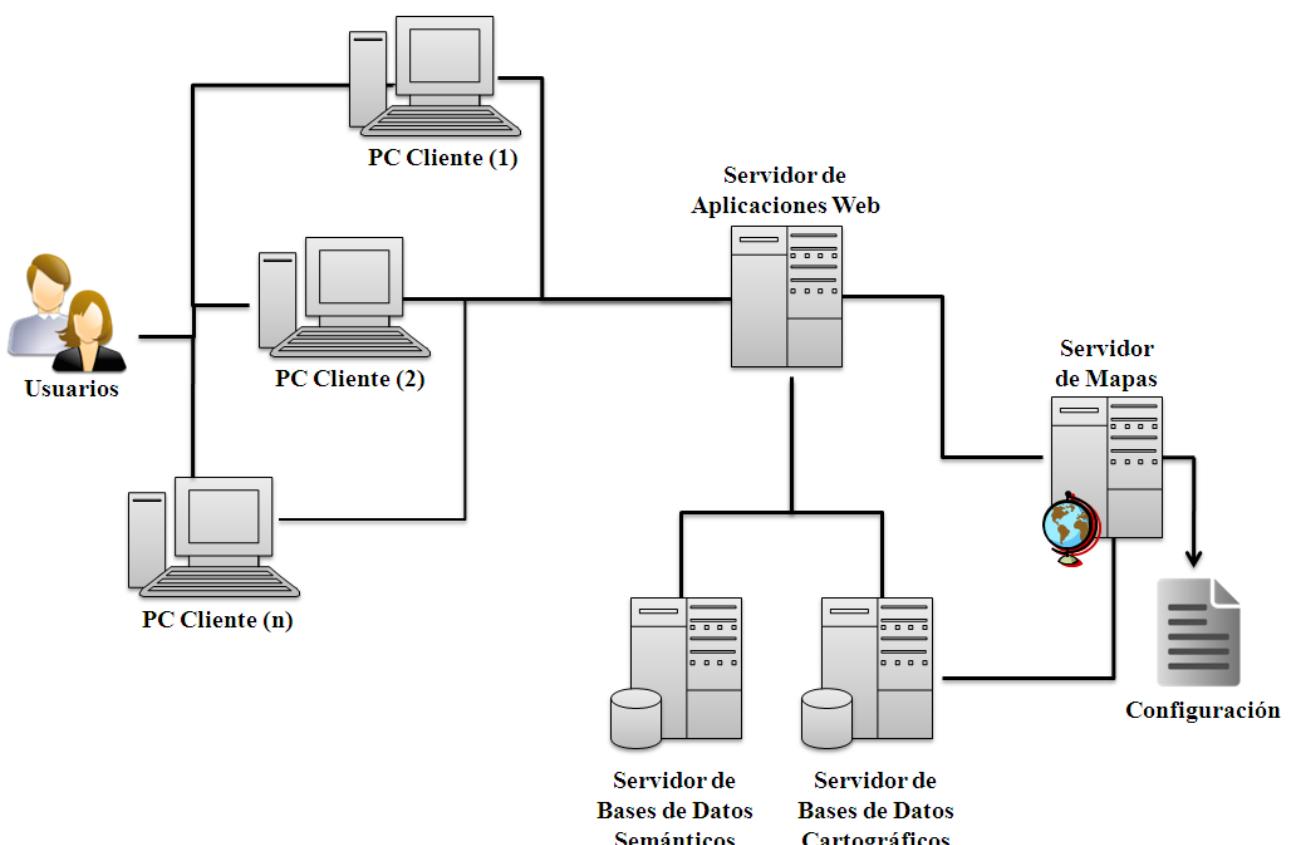


Ilustración 25 Entorno genérico para el despliegue de GeneSIG

Nótese que en este caso se incorpora un servidor dedicado a brindar servicios de mapas

digitales, el cual requiere elementos físicos de configuración como los *mapfile*, pero además este necesita consumir información que debe estar almacenada en una base de datos cartográfica, que por su complejidad se recomienda separar en un servidor independiente. Esta última a su vez puede también servir directamente al servidor de aplicaciones web.

5.1. Plugins y Coreplugins

Un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API. También se lo conoce como *plugin* (del inglés "enchufable"), *addon* (agregado), complemento, conector o extensión. La ventaja más notoria de los complementos es que permiten que los desarrolladores externos colaboren con la aplicación principal extendiendo sus funciones.

Partiendo de estos elementos se puede afirmar que un *plugin* constituye un módulo de *hardware* o *software* que añade una característica o un servicio específico a un sistema más grande. La idea es que el nuevo componente se acople simplemente al sistema existente, sin que este se vea afectado por su existencia.

En GeneSIG los *plugin* son paquetes modulares de archivos (clases PHP, plantillas HTML, imágenes y otros recursos) que se utilizan para llevar a cabo una acción especializada: el formato de mapa principal, la interfaz de las capas de la navegación, la exploración del mapa (zoom, paneo etc.), las consultas, la autenticación, interfaces de búsqueda y muchos más.

Por otra parte los *coreplugin* son considerados aquellos *plugins* de "bajo nivel" que tienen una implicación de carácter vertical afectando la mayoría de los componentes de la plataforma. Estos llevan a cabo acciones como la manipulación del tamaño del mapa, herramientas de navegación, la selección de capas. Son utilizados con frecuencia en muchas aplicaciones CartoWeb pueden incluirse en esta categoría. Están siempre disponibles y activados. Como resultado, otros *plugins* pueden interactuar con ellos.

A continuación se muestra la estructura de directorios definida para determinado *plugin*, asumiéndose que la plataforma GeneSIG fue descargada en el directorio: /var/www/genesig/

- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/client/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/server/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/common/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/templates/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/htdocs/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/htdocs/gfx/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/htdocs/js/
- /var/www/genesig/projects/< projectName >/plugins/< pluginName >/htdocs/css/

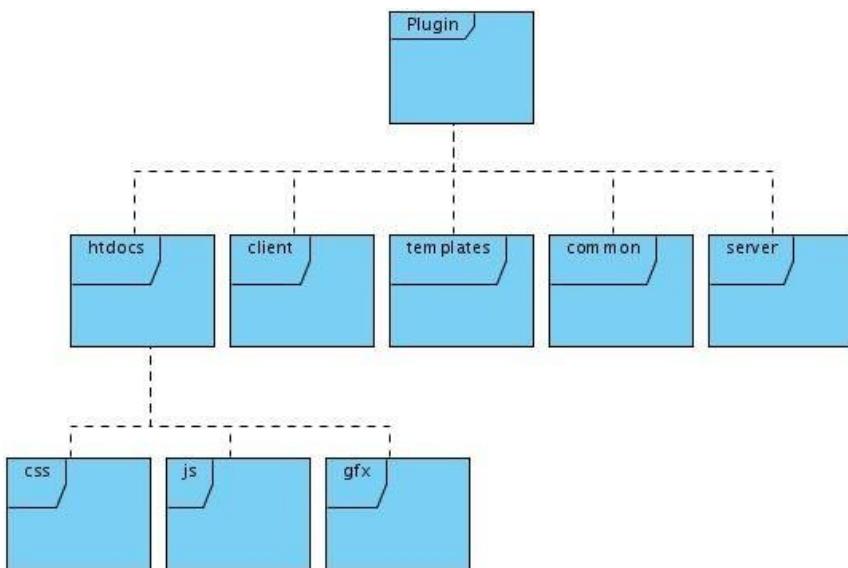


Ilustración 26 Estructura física para un *plugin* de GeneSIG

Breve descripción de los directorios:

- **client/** Contiene todos los componentes (de extensión *php*) del *plugin* que implementan el módulo *CartoClient* del Cartoweb.
- **server/** Contiene todos los componentes (de extensión *php*) del *plugin* que implementan el módulo *CartoServer* del Cartoweb.
- **common/** Contiene todos los componentes que implementan aquellas clases que servirán de puente para la comunicación entre *client.php* y *server.php*, incluyendo el fichero **.wsdl.inc*, donde se especifica como acceder a estas vía *SOAP*.
- **templates/** Contiene específicamente el fichero **.tpl* basado en el generador de plantillas Smarty para el procesamiento de datos.
- **htdocs/** Se define como el directorio que contiene los elementos que serán publicados por el servidor de aplicaciones *web*.
- **htdocs/gfx/** Contiene las imágenes que serán utilizadas en el *plugin*.
- **htdocs/css/** Contiene las hojas de estilo que serán utilizadas en el *plugin*.
- **htdocs/js/** Contiene todos los componentes (de extensión *js*) del *plugin* que implementan el módulo cliente en javascript del cartoweb.

5.2. Configuración

Teniendo en cuenta que se denomina configuración al conjunto de variables que controlan la operación general de un programa y que estas a su vez garantizan de forma transparente, la disposición de las partes más flexibles que componen dicho sistema en función de las necesidades del usuario. Se le da un tratamiento especial en GeneSIG y constituyen las bases para las personalizaciones de productos sobre esta plataforma. En la medida que se avance en el texto, se podrán apreciar un gran número de ficheros que sustenta dicha afirmación, aunque a grandes rasgos se agrupan en configuración de cliente y servidor.

5.2.1 Cliente

- **useWsdl:** Propiedad que define la posibilidad de emplear el modo *SOAP* o no, en dependencia del valor que la misma tome, los valores aceptados son: *true/false*

- **cartoclientBaseUrl**: dirección url del CartoClient
- **cartoserverBaseUrl**: dirección url del CartoServer
- **mapId**: propiedad de tipo cadena que define el nombre del proyecto concatenado con el nombre del mapa a utilizar a través de un carácter de tipo punto. Ejemplo se desea invocar el proyecto geoweb utilizando el mapa denominado cuba2001, entonces su mapid quedaría: `geoweb.cuba2001`
- **initialMapStateId**: propiedad de tipo cadena de caracteres, permite especificar el identificador del mapa a utilizar una vez iniciado el sistema, tomando este valor por defecto, sobre todo en caso de no ser especificado en la propia *url*.
- **loadPlugins**: Listado de los nombres que identifican a cada uno de los *plugin* que deban cargar su parte servidora cada vez que se solicita la ejecución de la plataforma.
- **smartyCompileCheck**: Propiedad que toma valores como `[true/false]`, se recomienda utilizar en falso durante las fases de desarrollo, permitiendo reconstruir las interfaces e identificar de esta forma los cambios así como los posibles errores

5.2.2 Servidor

En el modulo servidor existen claramente definidas una serie de variables de carácter configurativo que permiten controlar el comportamiento de la plataforma GeneSIG, a continuación se describen las principales.

- **imageUrl**: Toma valor de tipo *string* y describe la *url* mediante el cual se puede acceder a las imágenes generadas por *cartoserver*.
- **mapInfo.loadPlugins**: Listado de los identificadores asociados a aquellos *plugin* que requieran cargar la parte servidora.
- **showDevelMessages**: Propiedad de tipo booleana que define la salida de mensajes orientados a los desarrolladores, toma valores comprendidos en `[true/false]`.
- **developerIniConfig**: Propiedad de tipo booleana que define el paso de parámetros durante la inicialización, toma valores comprendidos en `[true/false]` y se utiliza mayormente en tiempo de desarrollo
- **noWsdlCache**: Propiedad de tipo booleana que define el empleo de cache para la generación del contrato de servicio en formato de wsdl.
- **noMapInfoCache**: Propiedad de tipo booleana que define el empleo de cache para las peticiones al método denominado *MapInfo*.
- **noMapResultCache**: Propiedad de tipo booleana que define el empleo de cache para las peticiones al método *getMap*.
- **noSoapXMLCache**: Propiedad de tipo booleana que define el empleo de cache para las peticiones *SOAP XML*

5.2.3 Adicionando módulos

Existen dos tipos de *plugin* en cartoweb (*coreplugin* y *plugin*), por tanto es muy importante tener esto en cuenta a la hora de implementar alguno, los *coreplugin* son aquellos que manipulan el sistema a bajo nivel, principalmente para el trabajo con los mapas, herramientas de búsqueda, selección de capas, etc y son cargados por defecto una vez que la aplicación comience a ejecutarse, generalmente son utilizados por los demás *plugin*, a pesar de esto la estructura interna de ambos es la misma y la gran diferencia radica en su configuración. En los ejemplos que se muestran a continuación se asume que el proyecto en construcción se denomina: `geoweb`.

- Para activar un **coreplugin** es necesario reimplementar los métodos encargados de invocar los módulos cliente y servidor una vez que la aplicación se comience a ejecutar, estos son *Cartoclient::getCorePluginNames()* y *ServerContext::getCorePluginNames()*.

Client path: /var/www/genesig/projects/<nombre del proyecto>/client/Cartoclient.php
Server path: /var/www/genesig/projects/<nombre del proyecto>/server/ServerContext.php

```
public function getCorePluginNames()
{
    $corepluginList = array('<nombre del nuevo coreplugin>');
    return array_merge(parent::getCorePluginNames(), $corepluginList);
}
```

En caso que se requiera cambiar tanto para el cliente como para el servidor, se podría modificar el contenido del fichero *Common.php* que se encuentra en el directorio /var/www/genesig/common/, aunque se recomienda que se realice a nivel de proyecto /var/www/genesig/projects/<nombre del proyecto>/common/, específicamente el método *Cartocommon::getCorePluginNames*.

- Para activar nuestro **plugin** es necesario configurar tres ficheros fundamentales:
 - client.ini**: Es el encargado de informarle al *cartoweb* que módulo cliente de plugis debe cargar cuando este se inicie, dicha información se debe especificar en la propiedad *loadPlugins*, es importante resaltar que si el *plugin* no consta de este modulo no se debe incluir, porque de ser así daría error.
Conf: loadPlugins = <nombre del nuevo plugin>
Path: /var/www/genesig/projects/<nombre del proyecto>/client_conf/client.ini
 - geoweb.ini**: Es el encargado de informarle al *cartoweb* que módulo servidor de plugis debe cargar cuando este se inicie, dicha información se debe especificar en la propiedad *mapInfo.loadPlugins*, es importante resaltar que si el *plugin* no consta de este modulo no se debe de incluir, porque de ser así daría error.
Conf: mapInfo.loadPlugin = <nombre del nuevo plugin>
Path: /var/www/genesig/projects/<nombre del proyecto>/server_conf/<nombre del mapa>/<nombre del mapa>.ini
 - cartoclient.tpl**: Es la plantilla principal del proyecto, en este fichero se debe especificar todo los js que se utilizarán en la aplicación.
Conf: <script type="text/javascript" src="../file.js"></script>
Path: /var/www/genesig/projects/<nombre del proyecto>/templates/cartoclient.tpl

A partir de la versión 1.5 de GeneSIG no es necesario modificar el tercer fichero pues se incluye una variable en los ficheros de configuración de cada modulo donde es posible especificar que js y css el mismo requiere para su invocación.

En la sección de vista denominada *views* se incluyen las variables que se describen a continuación:

- staticLoad**: Define el listado de ficheros escritos en lenguaje *Javascript* que serán cargados en cuanto se inicie la aplicación.
- dynamicLoad**: Define el listado de ficheros escritos en lenguaje *Javascript*

- que serán cargados bajo demanda.
- **cssLoad**: Define el listado de hojas de estilos que serán cargadas cuando se inicie la aplicación.
 - **compresJS**: Toma valores [0/1], determina se serán comprimidos los js que serán publicados, esta variante es recomendada para las etapas de despliegue ganándose por concepto de rendimiento.

Ejemplo del fichero de configuración para el modulo cliente del *plugin search*:

```
[views]
compresJS = 0
staticLoad[] = search.ajax.js

dynamicLoad[] = ejchart/dist/EJSChart.js
dynamicLoad[] = ejchart/JSCChart.js
dynamicLoad[] = search.aux.js
dynamicLoad[] = search.gui.js
dynamicLoad[] = search.tool.js

cssLoad[] = ./js/ejchart/dist/EJSChart.css
cssLoad[] = style.css

[Menu]
demo.id = "demo"
demo.name = "demo"
demo.text = "Demo"

demo.calculo.id = "democalculoid"
demo.calculo.name = "democalculoname"
demo.calculo.text = "Busqueda Sobre el Terreno"
demo.calculo.iconCls = "search-ico"
```

Si se observa detenidamente el ejemplo anterior se podrá apreciar la existencia de una sección denominada *Menu*. Esta tiene el objetivo de describir de la misma forma que se definen objetos en *Javascript* la estructura arboleara de un menú contextual, el cual sería incluido en la interfaz principal una vez cargado el modulo.

5.2.4 Accediendo a las variables

Independiente del tipo de *plugin* pueden tener tanto para el modulo servidor como el cliente, un fichero de configuración en el cual se pueden inicializar las variables que sean necesarias.

```
Client Path: /var/www/genesig/projects/<nombre del proyecto>/client_conf/<plugin_name>.ini
Server Path: /var/www/genesig/projects/<nombre del proyecto>/server_conf/<nombre del mapa>/<plugin_name>.ini
```

Para acceder a las variables definidas en los ficheros ".ini", se utiliza la llamada a la función *ClientPlugin::getConfig()* o *ServerPlugin::getConfig()*, en dependencia del modulo en que se esté programando, la sintaxis es:

```
$resultado = $this->getConfig()-><nombre de la variable>;
```

5.2.4 Variables de Entorno

GeneSIG define un conjunto de variables de solo lectura accesibles desde cualquier parte

del código fuente, las cuales provee información sobre el camino físico del núcleo de la plataforma, el proyecto, incluso el mapa en utilización.

- **CARTOWEB_HOME**: Dirección estática del código fuente de la plataforma GeneSIG, por ejemplo, si se asume que dicho producto fue descargado en el directorio: `/var/www/genesig/`, entonces ese sería el valor que tomaría esta constante.
- **CARTOWEB_PROY**: Ruta estática hacia el directorio del proyecto en ejecución, por ejemplo, si se asume que dicho producto fue descargado en el directorio: `/var/www/genesig/` y se está ejecutando el proyecto *herramienta*, seria: `/var/www/genesig/projects/herramienta/`
- **CARTOWEB_MAP**: Ruta estática hacia el directorio del mapa en ejecución dentro de un proyecto determinado, por ejemplo, si se asume que dicho producto fue descargado en el directorio: `/var/www/genesig/` y se está ejecutando el proyecto *herramienta* con el mapa *Cuba_250000*, seria: `/var/www/genesig/project/herramienta/server_conf/Cuba_250000/`
- **CARTOCOMMON_HOME**: Dirección estática de las clases responsables de controlar la información proveniente de los clientes, partiendo del mismo valor definido en el ejemplo para CARTOWEB_HOME, esta constante seria igual a `/var/www/genesig/common/`.
- **CARTOSERVER_HOME**: Dirección estática de las clases responsables de controlar la información proveniente de los clientes, partiendo del mismo valor definido en el ejemplo para CARTOWEB_HOME, esta constante seria igual a `/var/www/genesig/server/`.
- **CARTOCLIENT_HOME**: Dirección estática de las clases responsables de controlar la información proveniente de los clientes, partiendo del mismo valor definido en el ejemplo para CARTOWEB_HOME, esta constante seria igual a `/var/www/genesig/client/`.

5.3 Infraestructura

En este acápite se describe la estructura física que compone la plataforma GeneSIG, en principio es la misma definida por el *framework* CartoWeb.

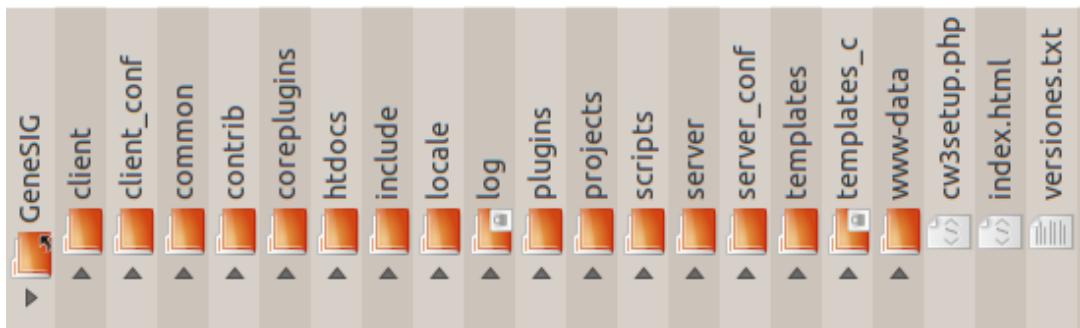


Ilustración 27 Taxonomía de la plataforma

Nótese la analogía que existe entre la estructura física de un *plugin* o *coreplugin* con un proyecto específico y a su vez con el *framework* en sentido general. Evidentemente es algo intencionado, en función de minimizar el cúmulo de conceptos que intervienen en el desarrollo de software sobre esta plataforma. A continuación se describen cada uno de

estos elementos:

- **client:** Directorio dedicado para almacenar las clases bases, responsables de la comunicación *cliente-servidor*, tales como: *AjaxHelper*, *ClientPlugin*, *Cartoclient*, *CartoserverService*, *ClientPluginHelper*, *HttpRequestHandler*, *ClientProjectHandler*, *ExportPlugin*, *FormRenderer*, *Smarty_Cartoclient*, etc.
- **client_config:** Directorio en que se alojan los ficheros de configuracion de la parte cliente.
- **common:** Directorio en que se almacenan las clases bases que constituyen el soporte arquitectónico tanto para la parte cliente como servidor, tales como *CartoLight*, *Log4phpInit*, *Message*, *ParseadorSQL*, *PluginBase*, *PluginManager*, *SecurityManager*, *PostgisDataGenesig*, *ProjectHandler*, etc.
- **contrib:** Directorio donde se almacenan algunas librerías necesarias para la plataforma tales como: *libParserWKT*, *pgdijkstra*
- **coreplugins:** Directorio donde se almacenan los *coreplugin* o componentes que tienen influencia directa y de carácter horizaontal para el resto de la plataforma.
- **htdocs:** Directorio de publicación, en el cual se encuentra ficheros tales como: *index.php*, *cartoserver.wsdl.php*, *client.php*, etc.
- **include:** Directorio en que son alojadas las librerias externas a la plataforma tales como: *CodeCompresor*, *ExtJs*, *FPDF*, *log4php*, *Smarty*, *DSN*, *PEAR*, *Yamasoft*, etc.
- **locate:** Directorio que contiene ficheros locales productos de la instalacion del sistema.
- **log:** Directorio en que son alojados los diferentes mensajes que permiten tener un control y trazabilidad de errores.
- **plugins:** Directorio en que se encuentran los plugin de carácter general y que pueden se reutilizados en varios proyectos.
- **projects:** Directorio en que se almacenan los distintos proyectos o aplicativos dirigidos a negocios específicos de un entorno determinado.
- **scripts:** Directorio que contienen disimiles ficheros *script* relacionados con la intalacion del marco de trabajo.
- **server:** Directorio dedicado para almacenar las clases bases, responsables de la interaccion con el servidor de mapas y publicación de servicios, tales como: *Cartoserver*, *cartoserver.wsdl*, *ServerPlugin*, *MapResultCache*, *ServerContext*, *LayerExt*, *SoapXMLCache*, etc.
- **server_conf:** Directorio en que se alojan los ficheros de configuracion de la parte servidora.
- **templates:** Directorio en que se almacenan las plantillas de propósito general, las cuales pueden ser redefinidas por los distintos proyectos.
- **cw3setup.php:** Fichero script que contiene el listado de instrucciones orientadas a la instalación y administración de los recursos de la plataforma.

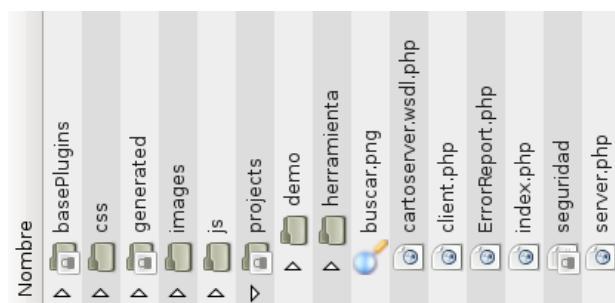


Ilustración 28 Estructura del directorio *htdocs* de GeneSIG

Un elemento importante a resaltar es el directorio *htdocs*, en el cual se publican los *scripts* que son entregados a los navegadores web. Como bien se puede observar en la figura anterior son generadas tres carpetas las cuales se describen a continuación:

- **basePlugins:** Directorio en que se encuentra el contenido de la carpeta *htdocs* de cada uno de los *plugins* bases que son utilizados por determinado proyecto. Esto es posible gracias al mecanismo de publicación de la plataforma el cual se activa una vez que se inicia la ejecución de un proyecto por parte del cliente.
- **generated:** Directorio en el cual se alojan aquellos archivos de carácter opcional para la plataforma, tales como la imágenes del mapa principal y de referencia, escalas, pdf, etc.
- **projects:** Directorio en el cual se almacena por proyecto, el contenido de la carpeta *htdocs* definida para cada uno de los plugin internos.

5.4 Interfaces

Teniendo en cuenta que el empleo de interfaces es uno de los recursos de la programación orientada a objetos que permite definir prototipos para ser aplicados a determinada jerarquía de clases, permitiendo el tratamiento de las instancias sin necesidad de conocer las características internas de las mismas., así como la complejidad de incorporar comportamientos a los *plugin* sin importar que tan heterogéneos estos sean. Se decide utilizar en GeneSIG esta técnica en función de garantizar los elementos básicos tales como la comunicación cliente-servidor, tratamiento de sesiones, etc.

A continuación se muestran las interfaces definidas por el marco de trabajo tanto para la parte servidora como cliente.

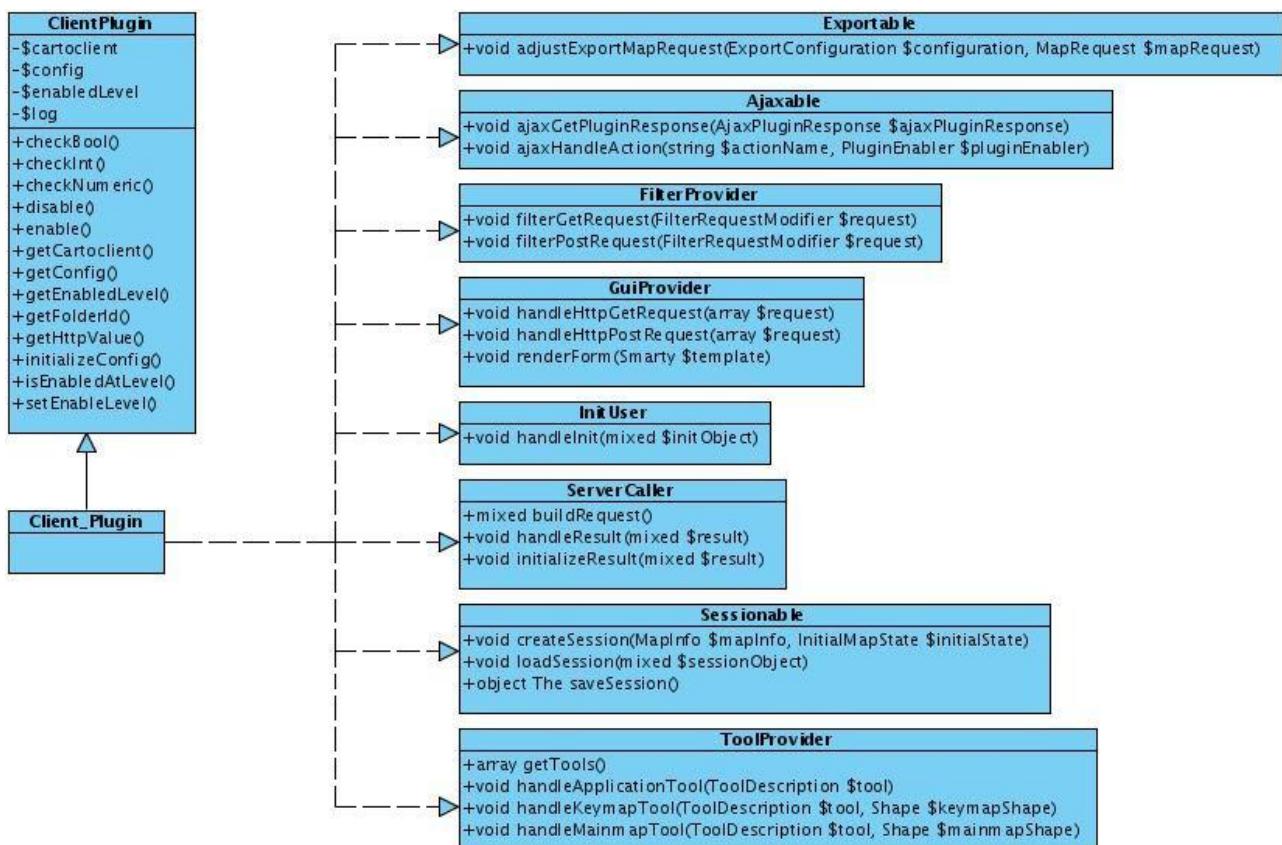


Ilustración 29 Interfaces definidas para el modulo cliente de un plugin o coreplugin

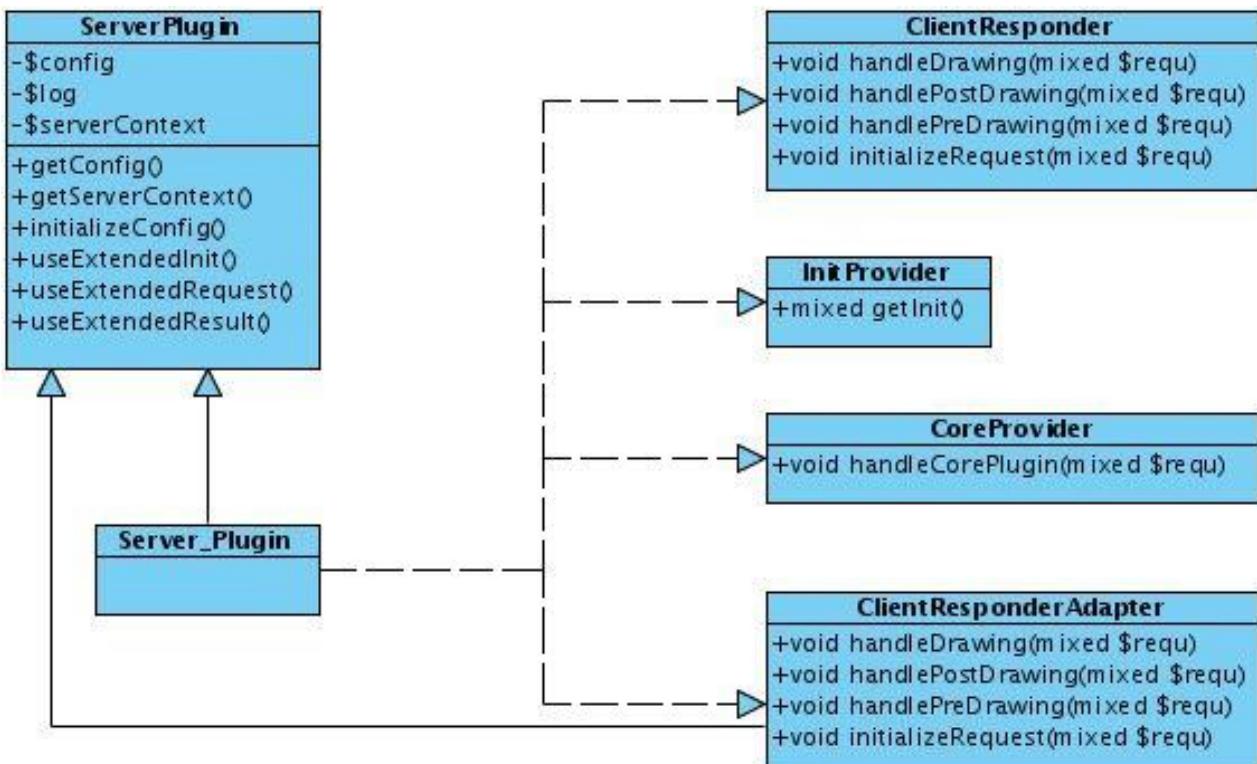


Ilustración 30 Interfaces definidas para el modulo servidor de un plugin o coreplugin

5.4.1 Comunicación

En la creación de un *plugin* este tipo de interfaces juegan un papel fundamental, pues proveen a los diferentes módulos de los mecanismos de comunicación entre ellos. Entre las principales que deben ser implementadas en el módulo cliente, se encuentran:

- **GuiProvider**: Para la comunicación con Smarty
- **ServerCaller**: Para la comunicación con el módulo Servidor
- **Ajaxable**: Para la comunicación con Ajax

En la parte servidora basta con extender de *ClientResponderAdapter*, pues esta implementa al resto de las interfaces, aunque en función de ganar en rendimiento se podrían definir de forma explícita la que sean requeridas. Para mayor comprensión de la utilización de estas clases anteriormente mencionadas refiérase al acápite 5.5. *Mecanismos de Comunicación*

5.4.2 Control de sesiones

La necesidad de las sesiones surge por la naturaleza del protocolo *HTTP*, el cual no dispone de un método incorporado para conservar el estado entre dos transacciones. Esto es lo que provoca que cuando un usuario solicita una página y luego otra, no exista manera de que el servidor entienda que ambas solicitudes provienen de un mismo usuario, de modo que todas las variables de un *script* son reestablecidas siempre después de una solicitud.

De forma simple, se puede definir una sesión como el tiempo que un usuario permanece conectado a un sitio web. De forma más técnica y relacionada con la programación del lado del servidor, una sesión es un bloque de información que almacena todo tipo de variables y valores relacionados con los usuarios y sus visitas a un sitio web en particular. El control de la sesión consiste en poder realizar un seguimiento del usuario mientras se

mantenga navegando por el sitio web, permitiendo mostrar contenido de las páginas en función de su nivel de autorización o de sus preferencias personales

Teniendo en cuenta la relevancia de lo anteriormente expuesto se decide incorporar un mecanismo para la gestión de sesiones de usuario a nivel de *plugin*. Para ello se define una interfaz en la parte cliente denominada *Sessionable*. La cual establece tres métodos que deben ser implementados:

- La función *Sessionable::createSession* recibe por parámetro dos objetos correspondientes a las clases *MapInfo* y *InitialMapState* respectivamente. Se invoca en el momento de creación de la sesión, generalmente se tiende a instanciar el objeto que encapsula las propiedades que serán almacenadas en sesión correspondiente al *plugin* en cuestión.
- La función *Sessionable::saveSession* debe retornar la estructura de dato que encapsule las propiedades que se requieran almacenar en sesión, en función del *plugin*.
- La función *Sessionable::loadSession* recibe por parámetro una instancia previamente almacenada en sesión, de la estructura definida para encapsular las propiedades que se requiera garantizar su persistencia, en función del funcionamiento del *plugin*.

A continuación se muestra un fragmento del *coreplugin* para la gestión de capas sobre el mapa denominado Layer, en el cual se ilustra su empleo:

```
class layersState
{
    public $classChecked;
    public $layersChecked;
    public $nodesExpanded;
    public $loaded;

    public function __construct()
    {
        $this->classChecked = array();
        $this->layersChecked = array();
        $this->nodesExpanded = array();
        $this->loaded     = false;
    }
}

class ClientLayers
    extends ClientPlugin
    implements Sessionable, GuiProvider, ServerCaller, Ajaxable
{

    public function loadSession($sessionObject) {
        $this->layersState = $sessionObject;
    }

    public function createSession(MapInfo $mapInfo, InitialMapState $initialMapState) {
        $this->locationState = new LocationState();
        $this->locationState->bbox = $initialMapState->location->bbox;
        $this->layersState = new layersState();
    }

    public function saveSession() {
        return $this->layersState;
    }

    public function initializeResult($result) {
        if ($result) {
            $this->layersState->classChecked = $this->classChecked;
            $this->layersState->layersChecked      = $this->layersChecked;
        }
    }
}
```

```
        $this->layersState->nodesExpanded = $this->nodesExpanded;
        $this->layersState->loaded = true;
    }
}
```

Nótese como en este caso se define la clase denominada *layersState* como contenedora de los atributos *classChecked*, *layersChecked*, *nodesExpanded*. Los cuales serán almacenados en sesión y se les introducirá información una vez que sea procesada la respuesta del servidor.

Con el fin de ganar en cuanto a la persistencia de los datos almacenados en sesión, se decide incorporar un mecanismo para gestión de cache, dirigidos a esta temática. Modificándose para ello el fichero *Cartoclient.php* ubicado en el directorio *client* de la raíz del marco de trabajo. Específicamente se le adiciona a la clase *ClientSession* la función denominada *saveCache*, la cual permite almacenar dicha información en el directorio definido para almacenar los datos asociados al perfil cartográfico por usuario. Esta se invoca posteriormente a la ejecución del comportamiento *saveSession*, definido en la función *Cartoclient::saveSession*, tal y como se muestra en el fragmento de código perteneciente a la clase *Cartoclient* que aparece a continuación.

```
public function saveSession()
{
    if ($this->preventSaveSession) {
        $this->log->debug('session save bypassed');
        return;
    }

    $this->callEnabledPluginsImplementing(
        ClientPlugin::ENABLE_LEVEL_PROCESS,
        'Sessionable',
        'saveSession'
    );

    $_SESSION[$this->getSessionName()] = $this->clientSession;

    session_write_close();
    $this->clientSession->saveCache( );
}

if ($this->areViewsEnable() && $this->isNewSession &&
!file_exists($this->getViewManager()->getSessionCacheLocation())) {

    $this->getViewManager()
    ->makeSessionCache($_SESSION[$this->getSessionName()])
}
}
```

De igual forma se implementa la función `ClientSession::loadCache`, permitiendo como su nombre lo indica cargar los datos asociados al usuario en cuestión, siendo invocada previamente a la ejecución de los comportamientos (`loadSession` y/o `createSession`) definidos en `Cartoclient::initializeSession`.

5.5. Mecanismos de Comunicación

Como bien se había mencionado en acápitos anteriores, la plataforma GeneSIG no presenta un comportamiento tradicional, comparado con otros *framework* orientados al desarrollo de aplicaciones web. A diferencia de los demás este marco de trabajo realiza una secuencia de acciones por petición, en función de las distintas interfaces que se hallan implementado tanto en la parte servidora como cliente de los plugin activos. A continuación se muestra una imagen que ilustra este mecanismo.

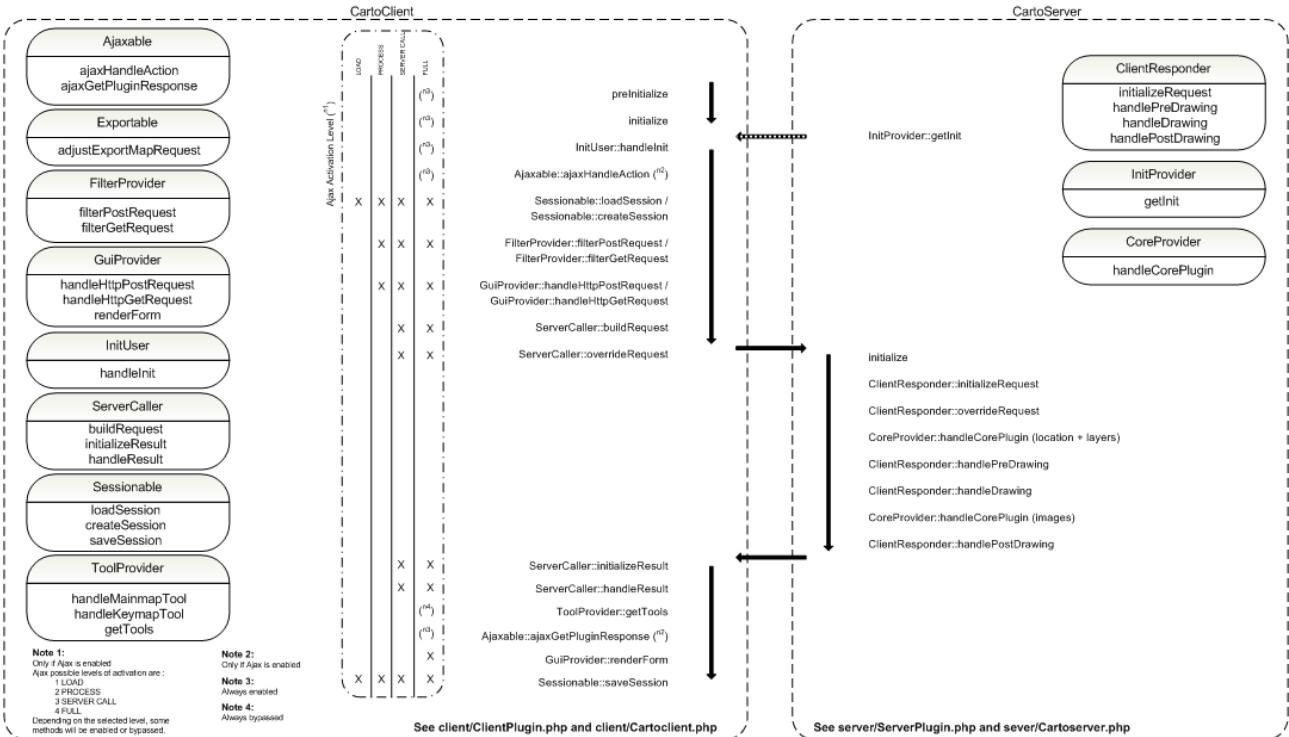


Ilustración 31 Secuencia de invocación de eventos por interfaces

5.5.1 Comunicación Client.php con Smarty

Smarty es un motor de plantillas para PHP, con la responsabilidad de separar el código escrito en dicho lenguaje como lógica de negocios, el código HTML como lógica de presentación y generar contenidos web mediante la colocación de etiquetas de tipo smarty en un documento. Teniendo en cuenta sus características y ventajas la comunidad de desarrollo de CartoWeb decidió utilizarlo para el trabajo con vistas y por consiguiente GeneSIG lo hereda.

Como bien se había expresado anteriormente las plantillas contienen código HTML y un seudocódigo definido por el propio motor de plantillas, mediante el cual se definen las variables contenedoras de información, a continuación se muestra un fragmento de cómo debería quedar:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset={$charset}">
    <title>${titleGeneSIG} ${version}</title>
    <link rel="SHORTCUT ICON" href="{$GENESIG_FAVICON}">
  </head>
  ...
</html>
```

Nótese que entre los caracteres de {} se declaran con la misma síntesis que en el lenguaje *php*, aquellas variables contenedoras de información, provenientes del *client.php*. Otro elemento importante es que en caso que se requiera adicionar un fragmento de código *css* o *javascript*, debe ser a través de la etiqueta *{literal}* de lo contrario se produciría un resultado no esperado, a continuación se muestra un ejemplo.

```
<style type="text/css">
  {literal} .authuser-login {
    background-image: url(..plugins/auth/htdocs/gfx/user.png) !important;
  } {/literal}
```

```

</style>

<script type="text/javascript">
    {literal}
        var mybox = new Ext.BoxComponent({
            cls: 'login-separator',
            autoEl: {
                tag: 'div'
            },
            style : {
                margin : '5px 0px 0px'
            }
        });
    {/literal}
</script>

```

Para enviar valores a las variables declaradas en la *tpl* local del *plugin*, se debe de crear en el modulo cliente, un objeto de tipo *smarty* de la siguiente forma:

```
$smarty = new Smarty_Plugin( $this->getCartoclient(), $this );
```

Una vez creado, se debe emplear la función *assign* pasando por parámetro el nombre de la variable de *smarty* así como el valor que le será asignado tal y como se muestra a continuación:

```
$smarty->assign( 'titleGeneSIG', $valor );
```

También es posible incrustar todo el código que se encuentra en la *tpl* local del *plugin* en la *tpl* general del proyecto, para ello se debe redefinir la función GuiProvider::renderForm() que recibe por parámetro el objeto *smarty* de la *tpl* general del proyecto, el procedimiento es similar al anterior, pero en este caso el valor es el objeto de la *tpl* local.

```
$tpl_local_plugin = $smarty->fetch( 'perfil.tpl' );
$tpl_general_proyect->assign( "mylocalspace", $tpl_local_plugin );
```

En sentido general este recurso puede ser utilizado pues que es uno de los elementos que provee CartoWeb para la comunicación cliente servidor, sin embargo a partir de la versión 1.5 de GeneSIG prácticamente queda obsoleto, pues todo se basa en Ajax en aras de ganar por concepto de rendimiento, evitándose de esta forma la reconstrucción de la aplicación cada vez que se requiera una respuesta del servidor.

5.5.2 Comunicación Client.js con Client.php

La comunicación entre el navegador o la parte cliente del sistema que debe estar implementada en lenguaje Javascript y HTML, con la parte servidora desarrollada en PHP, se establece utilizando como base la tecnología Ajax. En función de lograr esto se crea por cada *plugin* un objeto dentro del espacio de nombres definido por GeneSIG denominado AjaxPlugins, con una restricción de nomenclatura que define como identificador del *plugin* el propio nombre especificando la primera letra en mayúscula. Por ejemplo en caso que se desee desarrollar un *plugin* denominado *search* se debería declarar de la siguiente forma:

```
AjaxPlugins.Search = {
    ...
}
```

Para materializar el mecanismo de envío se deben especificar un listado de acciones con al menos un elemento como es evidente, el atributo establecido para tal responsabilidad

se denomina *Actions* y por cada acción se deben especificar los métodos *buildPostRequest* y *onAfterAjaxCall*, aunque este último es de carácter opcional.

```
AjaxPlugins.Search = {
    Actions: {
        Perform: {
            buildPostRequest: function(argObject)
            {
                ...
            },
            onAfterAjaxCall: function(argObject, pluginOutput){
                ...
            }
        }
    }
}
```

El método *buildPostRequest* es invocado a través de un disparado definido por el API cliente de CartoWeb para mayor comprensión refiérase al acápite 5.6. *Controladores Frontales*. Esta función debe retornar una cadena en el formato establecido para peticiones GET sobre el protocolo HTTP, en la cual se define como & seguido del nombre de la variable, a continuación el símbolo de = junto al valor que tomaría dicha variable, por ejemplo, si se desean enviar dos variables al servidor una con un valor de texto “tusa” e identificador *name* y la otra con valor numérico 115 e identificador *age*, seria de la siguiente forma:

```
&name=tusa&age=115
```

Generalmente se necesita enviar estructuras de datos un tanto más complejas en dependencia del negocio que se esté gestionando, en este caso se recomienda codificarlo en formato JSON. De esta forma se gana en cuanto a limpieza y claridad de los datos, ejemplo se desea enviar un objeto de tipo persona con su información básica, desde la acción denominada *Perform*, la sintaxis seria la que se muestra a continuación.

```
Perform: {
    buildPostRequest: function(argObject)
    {
        return '&persona=' + JSON.stringify(argObject);
    }
}
```

En este caso se recomienda utilizar alguna librería que implemente la codificación a JSON, en el caso de GeneSIG utiliza ExtJs podría emplearse dicho recurso. Sin embargo nada de esto debería ser un problema teniendo en cuenta que GeneSIG provee una funcionalidad denominada *AjaxHelper.ToQueryString*, que abstrae a los desarrolladores de esta problemática, por tanto una variante al ejemplo anterior seria de la siguiente forma:

```
Perform: {
    buildPostRequest: function(argObject)
    {
        var obj = {
            name: "tusa",
            age: 115
        }
        return AjaxHelper.ToQueryString("persona", obj)
            + AjaxHelper.ToQueryString("info", false);
    }
}
```

Pomo se podrá apreciar en el ejemplo anterior *AjaxHelper.ToQueryString* recibe dos

parámetros uno sería el identificador de la variable y el segundo el valor que tomaría la misma. Por defecto la función está diseñada para codificar a JSON el valor en caso de que este sea una estructura de dato.

Obsérvese que la función *buildPostRequest* recibe un parámetro de carácter opcional, que en el ejemplo anterior se denominó *argObject*, este es un objeto que se pasa por parámetro en el momento de activar el disparador para dicho *ajaxaction*, para mayor comprensión refiérase al acápite 5.6. *Controladores Frontales*. Hasta el momento se ha visto como mandar información puntual desde un *ajaxaction* para determinado *plugin*, pero también es posible sin importar que *plugin* invoque la petición al servidor enviar un cumulo determinado de datos, para ello se puede emplear la función *AjaxHandler.buildPostRequest*, la cual se encarga de extraer la información contenida en el formulario general, que debe estar definido en la plantilla principal del proyecto.

```
Perform: {
    buildPostRequest: function(argObject)
    {
        var obj = {
            name: "tusa",
            age: 115
        }
        return AjaxHelper.ToQueryString("persona", obj)
            + AjaxHelper.ToQueryString("info", argObject.info)
            + AjaxHelper.ToQueryString("out", argObject.out)
            + AjaxHandler.buildPostRequest();
    }
}
```

Una vez que se halla ejecutado el *ajaxaction*, es posible capturar los datos enviados en la parte cliente del *plugin* implementada en lenguaje PHP, para ello es necesario implementar la interfaz *GuiProvider* y *Ajaxable* en caso que se requiera retornar una respuesta, para más información refiérase al acápite 5.4 *Interfaces*. En función de capturar estos datos es necesario redefinir los métodos *GuiProvider::handleHttpPostRequest* y *GuiProvider::handleHttpGetRequest* en dependencia del método de envío seleccionado tal y como se muestra en el ejemplo que aparece a continuación:

```
<?php
class ClientSearch extends ClientPlugin implements GuiProvider
{
    public function handleHttpPostRequest($request)
    {
        if(isset($request['persona']))
        {
            $this->persona = $request['persona'];
        }
    }

    public function handleHttpGetRequest($request)
    {
        $this->info = $request['info'];
    }
}
?>
```

Obsérvese que la información se almacena en una variable global a la clase a través del recurso *\$this->*, pues es la única variante de acceder a dicha información en otras partes del código, teniendo en cuenta como se había explicado en acápitones anteriores, a diferencia de los *framework* tradicionales en GeneSIG se ejecuta una secuencia de operaciones por cada petición que se le realiza al servidor. Por otra parte el problema con

el ejemplo anterior está en el caso que se requiera capturar el *request* sin importar que método de envío fuera utilizado, en este caso serían GET o POST pues son los únicos soportados.

```
<?php
class ClientSearch extends ClientPlugin implements GuiProvider
{
    public function handleHttpRequest($request)
    {
        if(isset($request['persona']))
        {
            $this->persona = $request['persona'];
        }
        if(isset($request['info'])) $this->info = $request['info'];
    }

    public function handleHttpPostRequest($request)
    {
        $this->handleHttpRequest($request);
    }

    public function handleHttpGetRequest($request)
    {
        $this->handleHttpRequest($request);
    }
}
?>
```

Para resolver esta problemática comúnmente se crea una función que sea invocada los las anteriores y recibiendo dicho parámetro, tal y como se muestra en el ejemplo anterior. En el caso del controlador ligero se resuelve de una forma más simple pues se le incorpora este comportamiento en la función denominada *handleRequest*, para mayor comprensión refiérase al acápite 5.6. *Controladores Frontales*.

En el caso que se requiera devolver una respuesta al servidor será necesario implementar la interfaz *Ajaxable*, específicamente implementar los métodos *Ajaxable::ajaxGetPluginResponse* y *Ajaxable::ajaxHandleAction*. El segundo asegura que se mantenga habilitada la respuesta Ajax de al menos el propio *plugin*. Para definir este comportamiento es de vital importancia que se analicen las dependencias entre los módulos porque se puede sobrecargar el servidor para determinada petición sin que sea realmente necesario. Por ejemplo, en caso de que se invoquen para un *plugin* nombrado *search*, dos funcionalidades que no tengan repercusión directa en la ejecución de otros módulos denominadas *Perform* y *Send*, se debería deshabilitar los restantes módulos principalmente los *coreplugin* ganándose de esta forma por concepto de rendimiento, y para el resto de las peticiones que si afectan o depende de otros módulos simplemente se le habilita su propia salida Ajax.

```
public function ajaxHandleAction($actionName, PluginEnabler $pluginsDirectives)
{
    switch ($actionName) {
        case 'Search.Perform':
            $pluginsDirectives->disableCoreplugins();
            $pluginsDirectives->enablePlugin('search');
        break;
        case 'Search.Send':
            $pluginsDirectives->disableCoreplugins();
            $pluginsDirectives->enablePlugin('search');
        break;
        default: $pluginsDirectives->enablePlugin('search'); break;
    }
}
```

Nótese que la función `Ajaxable::ajaxHandleAction` recibe dos, parámetros el primero contiene el nombre de la acción que fue invocada desde el navegador y el segundo contiene un objeto que permite controlar las directivas de ejecución de los *plugins*, las mismas son las que se reflejan a continuación:

1. **ClientPlugin::ENABLE_LEVEL_LOAD**: Se carga en memoria el código fuente del *plugin*.
2. **ClientPlugin::ENABLE_LEVEL_PROCESS**: Cumple con el nivel anterior y ejecuta las funcionalidades implementadas en el cliente.
3. **ClientPlugin::ENABLE_LEVEL_SERVERCALL**: Cumple con el nivel anterior y ejecuta las funcionalidades implementadas en el servidor.
4. **ClientPlugin::ENABLE_LEVEL_FULL**: Cumple con el nivel anterior y habilita la respuesta de GUI.

En caso que se requiera modificar estas directivas en aras de minimizar costos por concepto de rendimiento es posible acceder a las mismas a través de las siguientes funcionalidades:

- **ClientPlugin::setEnableLevel**: Establece el nivel de ejecución que recibe por parámetro.
- **ClientPlugin::enable**: Establece ENABLE_LEVEL_FULL como nivel de ejecución.
- **ClientPlugin::disable**: Establece ENABLE_LEVEL_LOAD como nivel de ejecución.
- **ClientPlugin::isEnabledAtLevel**: Permite identificar si un plugin determinado esta habilitado para determinado valor de tipo nivel de ejecución, el cual debe ser pasado por parámetro.

Puntualmente se pueden habilitar plugins a través del método `PluginEnabler::enablePlugin`, especificándose el identificador semántico del mismo. De igual forma se puede provocar el efecto contrario empleando `PluginEnabler::disablePlugin`. Otro de los recursos del parámetro es que permite deshabilitar todos los *plugin* a través de la llamada a la función `PluginEnabler::disablePlugins` o deshabilitar los *coreplugin* invocando el método `PluginEnabler::disableCoreplugins`, tal y como se muestra en el ejemplo anterior.

La función `Ajaxable::ajaxHandleAction` específicamente es la que determina que variables y con qué valores son enviados al receptor Ajax. Este método recibe por parámetro un objeto de tipo `AjaxPluginResponse` que permite incorporar dos tipos de variables en la salida, las cuales pueden ser accedidas desde el controlador frontal que implementa GeneSIG en el lenguaje Javascript. Ejemplo en caso que se deseé retornar como respuesta una estructura de datos, así como variables de tipos de datos nativos la sintaxis sería como la que se muestra a continuación:

```
public function ajaxGetPluginResponse(AjaxPluginResponse $ajaxPluginResponse)
{
    $out = array(
        array('lav'=>'int55', 'val'=>'55'),
        array('lav'=>'int23', 'val'=>'23'),
        array('lav'=>'int11', 'val'=>'11'),
        array('lav'=>'int56', 'val'=>'56'),
        array('lav'=>'int77', 'val'=>'77')
    );
    $out = json_encode($out);
    $ajaxPluginResponse->addHtmlCode('data', $out);
}
```

```

        $ajaxPluginResponse->addVariable( 'count', 2 );
    }
}

```

En caso de utilizarse el controlador ligero para realizar la petición, no sería necesario codificar a formato JSON la estructura de dato almacenada en la variable *out*, teniendo en cuenta que el mismo está diseñado para adaptar la salida según el formato solicitado. En este caso si se llega a omitir implicaría un error en la salida.

En el cliente implementado en lenguaje Javascript existen dos alternativas para capturar dicha respuesta. La primera y evidente sería la función denominada *onAfterAjaxCall* declarada en la propia definición del *action*, tal y como aparecería a continuación:

```

AjaxPlugins.Search = {
    Actions: {
        Perform: {
            buildPostRequest: function(argObject)
            {
                ...
            },
            onAfterAjaxCall: function(argObject, pluginOutput)
            {
                var obj = Ext.decode( pluginOutput.htmlCode['data'] );
                var info = pluginOutput.variables.count;

                alert( obj['val'] + " : " + info );
            }
        }
    }
}

```

Obsérvese que existen dos agrupaciones a través de las cuales se puede acceder a las variables enviadas desde el servidor *htmlCode* en caso de utilizarse la función *AjaxPluginResponse::addHtmlCode* y variables en caso de emplearse *AjaxPluginResponse::addVariable*.

Este caso descrito en el ejemplo anterior garantiza que únicamente para el *ajaxaction Perform* se ejecutará el receptor Ajax que el mismo describe, pero el problema estaría en caso que se requiera capturar dicha salida sin importar que acción fuera ejecutada. En función de solventar dicha problemática se puede declarar la función *handleResponse*, para mayor comprensión obsérvese el ejemplo que se muestra a continuación.

```

AjaxPlugins.Search = {
    handleResponse: function(pluginOutput){
        var info = pluginOutput.variables.count;
        alert( "Info to any action: " + info );
    },
    Actions: {
        Perform: {
            buildPostRequest: function(argObject)
            {
                return AjaxHandler.buildPostRequest();
            },
            onAfterAjaxCall: function(argObject, pluginOutput)
            {
                var obj = Ext.decode(pluginOutput.htmlCode['data']);
                var info = pluginOutput.variables.count;

                alert( obj['val'] + " : " + info );
            }
        }
    }
}

```

5.5.3 Comunicación Client.php con Server.php

Los modos de comunicación descritos en el acápite 5. Arquitectura se ponen de manifiesto en la comunicación cliente.php con servidor.php. Con el objetivo de materializar dicha interacción se debe implementar la interfaz denominada *ServerCaller*. De esta interfaz resaltar que la función que permite la ejecución de la parte servidora es *ServerCaller::buildRequest*, la misma debe retornar un valor distinto de 0 o false.

```
class ClientSearch extends ClientPlugin implements ServerCaller
{
    public function buildRequest()
    {
        if($this->myrequest)
        {
            $searchRequest = new SearchRequest();
            $searchRequest->factor = $this->factor;
            $searchRequest->coord = $this->coord;
            return $searchRequest;
        }else return 0;
    }
    public function initializeResult($searchResult){ ... }
    public function handleResult($searchResult) { ... }
}
```

Otro de los elementos claves es que para que funcione correctamente el modo SOAP debe implementarse dos clases como mínimo. Una que agrupe la información que será enviada al servidor denominada *SearchRequest*, en consecuencia con el ejemplo que se ha venido desarrollando en este material, aclarar que se aconseja definir el con la siguiente nomenclatura *<nombre del plugin>Request* y *<nombre del plugin>Result* para aquella que aglutina la respuesta generada por la parte servidora de dicho módulo, que en el caso de este ejemplo sería *SearchResult*. Estas clases deben estar definidas dentro del fichero *<nombre del plugin>.php* ubicado en el directorio: /var/www/genesig/projects/geoweb/plugins/search/common/

Cada una de estas clases contenedoras de información así como los tipos de datos de los elementos que las compongan, deben quedar declaradas en el fichero *<nombre del plugin en minúscula>.wsdl.inc* ubicado en el directorio /var/www/genesig/projects/geoweb/plugins/search/common/, cuya sintaxis para el ejemplo que se venía mostrando, es la que se muestra a continuación:

```
<complexType name="SearchRequest">
<all>
    <element name="className" type="xsd:string"/>
    <element name="factor" type="xsd:int"/>
    <element name="coord" type="types:string"/>
</all>
</complexType>
```

Un elemento importante a tener en cuenta, es que tanto *ClientRequest* como *ServerResult* deben heredar de la clase *CwSerializable* y redefinir el método *unserialize()*, con el objetivo de tipar cada una de las variables que estas tengan definidas, de lo contrario no se efectuaría la comunicación vía SOAP, en caso de especificarse alguno de los modos de comunicación que utilicen dicho protocolo, la sintaxis es la siguiente:

```
public function unserialize($struct)
{
    $this-><variable> = self::unserializeValue($struct, '<variable>', '<tipo de dato>');
}
```

Por otra parte el modulo servidor debe implementar alguna de las interfaces definidas para garantizar dicha comunicación, usualmente se utiliza *ClientResponderAdapter*, para mayor comprensión refiérase al capítulo 5.4 *Interfaces*.

```
class ServerSearch extends ClientResponderAdapter
{
    public function handlePreDrawing($requ)
    {
        $result = new SearchResult();
        $result->serialPunto = $this->serieP;
        $result->serialArea = $this->serieA;
        return $result;
    }
}
```

Nótese que la función *ClientResponderAdapter::handlePreDrawing* debe retornar un valor distinto de 0 o *false*, principalmente de un tipo de dato descrito para la plataforma como es el caso de *SearchResult*. Este resultado generado puede ser capturado en el modulo cliente a través de dos funciones principales *ServerCaller::initializeResult* y *ServerCaller::handleResult*.

```
class ClientSearch extends ClientPlugin implements ServerCaller
{
    public function buildRequest(){ ... }
    public function initializeResult($searchResult){ ... }
    public function handleResult($searchResult)
    {
        if (!empty($searchResult))
        {
            $this->serialPunto = $searchResult ->serialPunto;
            $this->serialArea = $searchResult ->serialArea;
        }
    }
}
```

Como se puede apreciar en el ejemplo, generalmente el resultado obtenido se almacena en variables globales a la clase, con el objetivo de realizar el envío de datos al navegador web o seguir procesando dicha información en otro de las funciones que intervienen en el flujo invocado. Para mayor comprensión refiérase al acápite 5.5.2 *Comunicación Client.js con Client.php*.

5.5.4 Comunicación entre plugins

Cartoweb provee un mecanismo de comunicación entre *plugin*, permitiendo que se puedan reutilizar los componentes que se hayan desarrollado de forma simple y sin necesidad de aumentar el acoplamiento entre las clases que intervienen en la solución. Accediendo desde la propiedad *ServerPlugin::serverContext* a la función *ServerContext::getPluginManager()* es posible obtener el listado de los objetos servidores de cada uno de los *plugin* que se encuentran activos, a continuación se puede observar la sintaxis:

```
$controladorPlugin = $this->serverContext->getPluginManager();
```

Como bien se había expresado anteriormente la variable *\$controladorPlugin* recibe un objeto de tipo *PluginManager* el cual permite acceder directamente a cada uno de los *plugin* activos, por ejemplo si se desea realizar alguna de las consultas que este definida en el modulo servidor del *coreplugin mapquery*, bastaría con:

```
$mapqueryPlugin = $controladorPlugin ->mapquery;
$Resultado_ids = $mapqueryPlugin ->queryByIdSelection($Selection, $Fail);
```

Desde el modulo cliente también es posible acceder a estos *plugins*, pero en este caso se accede desde la propiedad heredada *ClientPlugin::cartoclient*, la cual no es más que un objeto de la clase CartoClient, permitiendo obtener de esta forma el objeto controlador de *plugins*.

```
$controladorPlugin = $this->cartoclient->getPluginManager();
```

5.6 Controladores Frontales

El controlador frontal es considerado un patrón de diseño que se basa en usar un controlador como punto inicial para la gestión de las peticiones. Este gestiona las peticiones, y realiza algunas funciones como comprobación de restricciones de seguridad, manejo de errores, mapear y delegación de las peticiones a otros componentes de la aplicación que se encargarán de generar la vista adecuada para el usuario. Permitiendo centralizar en un único punto la gestión de las peticiones, aumentando el control sobre estas, así como la reusabilidad de código, mejorándose la gestión de la seguridad entre otros elementos. Aunque la velocidad de respuesta disminuye al tener que ser procesadas las peticiones primero por el controlador, sin embargo es prácticamente despreciable cuando se pone en una balanza de costo beneficio.

Pese a que CartoWeb presenta un comportamiento atípico con respecto a los *framework* tradicionales para el desarrollo de aplicaciones web también pone en práctica este precepto arquitectónico. Sin embargo por falta de uno presenta dos implementaciones, denominadas como controlador ligero y pesado, los cuales se describen a continuación.

Es importante destacar que todas las peticiones web son manejadas por un solo *script* frontal denominado *index*, el cual constituye el punto de entrada único de toda la plataforma en un entorno determinado y en dependencia de los parámetros este decide que controlador instanciar. Por defecto se asume como el controlador nativo el denominado *heavy* o pesado. Este es realmente la implementación original provista por el *framework* CartoWeb y por consiguiente carga en memoria todos los recursos implementados por dicho marco de trabajo. Además por cada petición se ejecutan de forma secuencial un conjunto de módulos, de ahí que en ocasiones resulte poco viable utilizar todo esto, en función de resolver algo muy particular y q requiera poco procesamiento.

CartoLight es la denominación que se le impone a una implementación del propio *framework* de CartoWeb. Esta recrea el flujo tradicional definido por su predecesor, con la peculiaridad de que no instancia la inmensa mayoría de los recursos del marco de trabajo, de ahí la idea de petición ligera. Este tipo de peticiones se caracteriza por cargar en memoria un solo *plugin* y aquellos elementos requerido por este, en función de garantizar el flujo básico de ejecución, como son los recursos comunes, de seguridad, configuración, tratamiento básico de excepciones, log, respuesta de Ajax y en el caso del servidor el *ServerContext*.

Independientemente de las interfaces implementadas, las llamadas a los procedimientos definidos tanto en la parte cliente como servidora del módulo, son las que se describen a continuación y en el mismo orden que aparecen.

1. Client::**handleHttpPostRequest(\$_POST)**
2. Client::**handleHttpGetRequest(\$_GET)**
3. Client::**handleRequest(\$_REQUEST)**

4. \$request = Client::buildRequest()
5. \$result = Server::handlePreDrawing(\$request)
6. Client::initializeResult(\$result)
7. Client::handleResult(\$result)
8. return Client::ajaxGetPluginResponse(\$ajaxPluginResponse)

Obsérvese que la tercera llamada denominada *handleRequest* se adiciona como elemento conceptual que agrupa las dos primeras, pero que no se encuentra definida en las interfaces oriundas de CartoWeb, esto implica mucho cuidado por parte de los desarrolladores en aras de garantizar la compatibilidad en cuanto al funcionamiento de nuestros módulos, sin importar que tipo de petición se esté realizando, dígase en modalidad pesada o ligera.

Los valores de entrada que caracterizan a este tipo de peticiones son los que se describen a continuación:

- **typeRequest:** Define la variante de petición a realizar, valores esperados (*light* | *heavy*), por defecto toma valor *heavy*
- **project:** Identificador del proyecto, este valor es necesario en caso de no especificar el mapid
- **typeOut:** Formato de salida, valores esperados (*xml* | *json*), valor por defecto toma valor *xml*
- **typeModule:** Define el tipo de modulo a ejecutar, valores esperados (*plugins* | *coreplugins*), sin embargo toma valor por defecto el valor de *plugins*
- **mapid:** Identificador del proyecto seguido por el identificador del mapa unidos por el carácter ".", ejemplo "[geoweb.geoweb](#)"
- **pluginName:** Nombre identificador del modulo a ejecutar

Otro elemento importante a mencionar es el relacionado con la respuesta del modulo. Generalmente cuando se realiza una petición ligera está estrechamente relacionada con la necesidad de actualizar algún elemento de interfaz grafica de usuario, en los que su mayoría requieren una estructura de datos compleja. Evidentemente lo más factible sería devolver dicha respuesta en formato *json*, sin embargo hay que tener mucho cuidado, pues generalmente los desarrolladores terminan la ejecución del *plugin* una vez terminado el procesamiento que estos requieren, a través de la función **die**, realizando estos mismos la codificación a dicho formato. Esto implica que asumen la responsabilidad del comportamiento de modulo en sí mismo, así como la calidad de la codificación y por otra parte estaría la limitación de posibilidad de brindar servicios web. En función de solventar y de compatibilizarlo con otros formatos, se define la propiedad *typeOut*, la cual permite obtener un resultado similar pero basado en flujo estándar definido por CartoWeb.

En caso que se desee ejecutar el *plugin* perfil a través del controlador ligero, definiéndose como directorio de publicación */localhost/GIS/Genesig/* y especificando los parámetros requeridos para llevar a cabo los cálculos necesarios y obtener el resultado en formato *json*, la url debería quedar de la siguiente forma:
[http://localhost/GIS/Genesig/htdocs/index.php?ajaxActionRequest=Perfil.Perform&project=herramienta &Factor=900&ListCoord=\[\[-82.0681301, 23.167474, 0\], \[-81.33789, 22.667817, 93275.74\], \[-77.6867, 21.51476, 491156.26\] \]](http://localhost/GIS/Genesig/htdocs/index.php?ajaxActionRequest=Perfil.Perform&pluginName=perfil&project=herramienta&typeModule=plugins&typeOut=json&typeRequest=light &Factor=900&ListCoord=[[-82.0681301, 23.167474, 0], [-81.33789, 22.667817, 93275.74], [-77.6867, 21.51476, 491156.26]])

Obsérvese la diferencia en caso de que se necesitara ejecutar esta *url* pero desde el controlador tradicional: [http://localhost/GIS/Genesig/htdocs/index.php?ajaxActionRequest=Perfil.Perform&project=herramienta &Factor=900&ListCoord=\[\[-82.0681301, 23.167474, 0\], \[-81.33789, 22.667817, 93275.74\], \[-77.6867, 21.51476, 491156.26\] \]](http://localhost/GIS/Genesig/htdocs/index.php?ajaxActionRequest=Perfil.Perform&project=herramienta &Factor=900&ListCoord=[[-82.0681301, 23.167474, 0], [-81.33789, 22.667817, 93275.74], [-77.6867, 21.51476, 491156.26]])

5.6.1 Generador de URL

Evidentemente conformar de forma manual estas *url* es un proceso tedioso, sobre todo si el factor humano se ve expuesto en cuanto a los errores de sintaxis que esto podría acarrear, por tal motivo el equipo de desarrollo incorporo al API cliente de GeneSIG un recurso que permite automatizarlo. Este recurso pertenece a la instancia *CartoWeb* y se denomina *getLightURL*, esta función recibe varios parámetros los cuales se describen continuación, en el mismo orden en que deben ser especificadas:

- **ajaxAction:** Valor compuesto en formato cadena que identifica el plugin y la acción invocada, ambos deben ser concatenados por un carácter de punto, ejemplo en el caso de la acción Perform del plugin Perfil seria: *Perfil.Perform*
- **params:** Estructura en forma de arreglo asociativo identificado por índices semánticos los cuales especifican los parámetros para la *url* generada, los cuales se describen a continuación:
 - **typeRequest:** Define la variante de petición a realizar, valores esperados (*light* | *heavy*), por defecto toma valor *heavy*
 - **project:** Identificador del proyecto, este valor es necesario en caso de no especificar el *mapid*
 - **typeOut:** Formato de salida, valores esperados (*xml* | *json*), valor por defecto toma valor *json*
 - **typeModule:** Define el tipo de modulo a ejecutar, valores esperados (*plugins* | *coreplugins*), sin embargo toma valor por defecto el valor de *plugins*
 - **mapid:** Identificador del proyecto seguido por el identificador del mapa unidos por el carácter ".", ejemplo "geoweb.geoweb"
 - **pluginName:** Nombre identificador del modulo a ejecutar

En caso de necesitar definir un *store* para algún componente de interfaz grafica de usuario, perteneciente al *plugin* de configuración, específicamente el *ajaxaction default*, quedaría de la siguiente forma:

```
AjaxPlugins.Configuracion.STAUser= new Ext.data.Store({  
    url: CartoWeb.getLightURL('Configuracion.default')+'&action=GetDGUsuario',  
    reader:new Ext.data.JsonReader({  
        totalProperty: "cantidad_filas",  
        root: "datos",  
        id: "ide"  
    },  
    [ {  
        name:'idaccion',  
        mapping:'idaccion'  
    },{  
        name:'denominacion',  
        mapping:'denominacion'  
    },{  
        name:'descripcion',  
        mapping:'descripcion'  
    }],  
    sortInfo:{  
        field: 'denominacion',  
        direction: "ASC"  
    }  
});
```

Nótese que se le adiciona una variable denominada *action* cuyo valor es *GetDGUsuario*, y al no especificar ningún otro parámetro se asumen los restantes valores por defecto.

5.6.2 Ejecutor de peticiones al servidor

Otro de los elementos necesario para desarrollar nuestros módulos consiste en conocer como invocar en determinadas condiciones una acción al servidor. En el apartado anterior se describe como se conformaría la *url* para que un controlador sea capaz de establecer dicha comunicación, sin embargo no siempre contamos con *store* u otros componentes que implementen este tipo de interacción. En consecuencia a esto GeneSIG provee dos recursos para establecer una comunicación entre cliente y servidor basada en la propia tecnología Ajax, permitiendo crear de esta forma un *buffer* de respuestas, las cuales podrán ser capturadas en los pertinentes controladores por modulo.

CartoWeb.trigger y *CartoWeb.triggerLight* constituyen los mecanismos rectores para invocar el desenlace de ejecuciones prevista por cada petición realizada al servidor. Estos pertenecen al *coreplugin* base implementados en el *dhtmlAPI.js*. El primero hace referencia al método tradicional o basado en el controlador pesado de CartoWeb y el segundo recrea este proceso para el controlador ligero definido por GeneSIG, ambos están constituidos por los mismos parámetros en aras de garantizar su compatibilidad, los cuales se describen a continuación:

- **ajaxAction:** Valor compuesto en formato cadena que identifica el *plugin* y la acción invocada, ambos deben ser concatenados por un carácter de punto, ejemplo en el caso de la acción *Perform* del *plugin Perfil* seria: *Perfil.Perform*
- **nonAjaxInstruction:** Cadena de caracteres que definen el conjunto de acciones a realizar en caso de que no se requiera utilizar el modo de Ajax, previamente especificado a través de *CartoWeb.disableAjax()* y *CartoWeb.enableAjax()* respectivamente, esta información se determina a través de la función *CartoWeb.isAjaxMode()*, como valor por defecto toma vacío.
- **ajaxArgObject:** Define un listado de argumentos que son pasados como parámetros en la acción Ajax definida por el *ajaxAction*.

Por ejemplo en caso de que se requiera invocar la acción *Perform* perteneciente al *plugin Perfil*, debería especificarse la llamada de la siguiente forma:

```
CartoWeb.trigger ("Perfil.Perform");
```

Nótese que se estaría ejecutando el controlador tradicional y se asume el resto de los valores por defecto. Lo que implica que si *CartoWeb.isAjaxMode()* retorna valor falso no habría nada, teniendo en cuenta que no se específico el conjunto de instrucciones para cuando no esté habilitado el empleo de la tecnología Ajax. Por consiguiente si se requiere que dicha ejecución sea a través del controlador ligero quedaría de la siguiente forma:

```
CartoWeb.triggerLight("Perfil.Perform", "alert('sorry no ajax mode');");
```

En caso de requerir información que varié en dependencia del contexto de invocación sería prudente utilizar el tercer parámetro, tal y como se muestra a continuación:

```
CartoWeb.triggerLight("Perfil.Perform", null, { factor:2500, lonlat: [-88.3535, 17.0505] });
```

Nótese que en este último caso se podría acceder a dichos parámetros desde la controladora definida para la parte cliente del *plugin Perfil* de la siguiente forma:

```
AjaxPlugins.Perfil = {
    handleResponse: function(pluginOutput){ ... },
    init: function(){ ... },
    Actions: {
        Perform: {
```

```

buildPostRequest: function(argObject)
{
    ...
    var request = AjaxHelper.ToQueryString("factor", argObject.factor);
    request += AjaxHelper.ToQueryString("lonlat", argObject.lonlat);
    return request;
},
onAfterAjaxCall: function(argObject){ ... }
}
};


```

5.7 Acceso a Datos

GeneSIG para la manipulación y control de los datos gestionados a través de un SGDB emplea PEAR ubicada en el directorio `./include/wrapperpear/` a partir de la propia raíz del marco de trabajo, para mayor compresión sobre esta librería refiérase al acápite 2.5 *PEAR*. A continuación se describen algunos de los mecanismos más relevantes relacionados con este tema incluyendo recursos adicionados por el propio equipo de desarrollo del marco de trabajo en función de extender sus posibilidades.

5.7.1 Ejecutar un script SQL

Tanto en el modulo cliente como en el servidor de cualquier *plugin*, se puede acceder a través de la librería PEAR a una base de datos externa, sin embargo teniendo en cuenta elementos como la responsabilidad que asumen cada uno de estos modulos se recomienda que sea en el modulo servidor del mismo. En función de esto se implementaron algunos recursos que facilitan el trabajo con esto, tal es el caso de la función `ServerPlugin::getConnectionString`, esta permite obtener un objeto de tipo conexión el cual permite una serie de funcionalidades las culés se describirán más adelante.

Independientemente de lo anteriormente expresado, se puede utilizar este recurso en cualquier parte de la plataforma tan solo especificando la dirección de la librería, tal y como se muestra a continuación.

```
require_once( CARTOWEB_HOME . 'include/wrapperpear/DB.php' );
```

Para utilizar `ServerPlugin::getConnectionString` se deben de especificar dos parámetros: *name* y *trigger*. El primero define un tipo de sobrecarga para esta función, permitiendo especificar tanto una conexión en formato *dsn*²⁷, como un simple identificador que debe estar especificado en el fichero `modulesconfig.xml`, definido en el directorio contenido en la variable global `CARTOWEB_MAP`, con una sintaxis muy similar a la que se muestra a continuación:

```

<config>
<conn formato="dsn" bd="Genesig" gestor="pgsql" host="10.12.170.140" port="5432" usuario="ing"
password="ingsig09Grm"/>

<DATOSSIG>
    <conn bd="DATOSSIG" usuario="ing" password="ingsig09Grm" />
</DATOSSIG>

<SIGGRANMA>
    <conn bd="SIGGRANMA2011" usuario="ing3" password="ingsig09Grm3" />
</SIGGRANMA>
</config>
```

²⁷ **dns** acrónimo de Data Source Name, del español Nombre Fuente de datos o Nombre de origen de datos.

Obsérvese que tanto para los identificadores de conexión: DATOSSIG y SIGGRANMA, así como el resto que se defina a continuación, presentan la peculiaridad de que aquellos atributos que no sean definidos en el *conn* propio lo asumen del que se encuentra a su mismo nivel. También esto puede ser solventado a través del índice denominado *xmlconexionBD* definido en el fichero de extencion *.ini* del servidor para determinado proyecto. Teniendo en cuenta esto, una forma de utilizarlo sería:

```
$this->db = $this->getConnectionTo('SIGGRANMA2011');
```

De esta forma estaríamos obteniendo una conexión a la base de datos *Genesig*, ubicada en el servidor *10.12.170.140*, con el usuario *ing3* y la contraseña *insig09Grm3*. Por otra parte podría obtener la misma especificándoselo de forma manual tal y como se muestra a continuación:

```
$host      = "10.12.170.140";
$usuario   = "ing3";
$password  = "insig09Grm3";
$dbname    = "Genesig";
$p puerto  = "5432";
$this->db   = $this->getConnectionTo("pgsql://$usuario:$password@$host/$dbname");
```

Al igual que en el caso anterior, este comportamiento constituye una capa de abstracción a dato pr encima de las funciones nativas del propio PEAR, equivalente a:

```
$this->db =& DB::connect('pgsql://$usuario:$password@$host/$dbname');
```

El segundo parámetro denominado *trigger* es de carácter booleano y asume por defecto valor verdadero, esto implica su capacidad de ejecutar la función *Utils::checkDbError*, la cual posibilita realizar el chequeo de errores asociados a bases de datos, deteniendo la ejecución o no del script.

Otro elemento que resulta de bastante utilidad es especificar el formato de la estructura resultante, a través de la función *setFetchMode* y especificando la opción *DB_FETCHMODE_ASSOC* es posible obtener un arreglo asociativo basado en índices semánticos, tal y como se muestra a continuación:

```
$this->db->setFetchMode(DB_FETCHMODE_ASSOC);
```

Una vez configurado los elementos necesarios en función de ejecutar la consulta sería tan simple como conformar dicha sintaxis *sql* e invocar la función *query* tal y como se muestra a continuación:

```
$result =& $this->db->query("SELECT * FROM clientes");
```

Como mecanismo de obtención de los datos se puede emplear la función *fetchInto* y a través de un *foreach* recorrer lista, para mayor compresión refiérase al ejemplo que aparece a continuación:

```
while ($result->fetchInto($row)) {
    echo $row[0] . ':' . $row[1] . "\n";
}
$db->free();
$db->disconnect();
```

Como resultado en este ejemplo se concluye liberando la memoria a través de la función *free* y cerrando el canal de conexión para con la bases de datos.

5.7.2 Escape SQL

Inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos. El origen de la vulnerabilidad radica en el incorrecto chequeo y/o filtrado de las variables utilizadas en un programa que contiene, o bien genera, código SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o script que esté embebido dentro de otro.

Por ejemplo, asumiendo que el siguiente código reside en un aplicativo desarrollado sobre GeneSIG y que existe un parámetro *userName* que contiene el nombre de usuario a consultar, una inyección SQL se podría provocar de la siguiente forma:

```
$query = "SELECT * FROM usuarios WHERE nombre = '" . $userName . "';"
```

Si el operador escribe un nombre, por ejemplo *alicia*, nada anormal sucederá, la aplicación generaría una sentencia SQL similar a la siguiente, que es perfectamente correcta, en donde se seleccionarían todos los registros con el nombre "alicia" en la base de datos:

```
SELECT * FROM usuarios WHERE nombre = 'alicia';
```

Pero si un operador malintencionado escribe como nombre de usuario a consultar: *'alicia'*; *DROP TABLE usuarios;* *SELECT * FROM datos WHERE nombre LIKE %;*, se generaría la siguiente consulta SQL:

```
SELECT * FROM usuarios WHERE nombre = 'alicia';
DROP TABLE usuarios;
SELECT * FROM datos WHERE nombre LIKE '%';
```

En la base de datos se ejecutaría la consulta en el orden dado, se seleccionarían todos los registros con el nombre *alicia*, se borraría la tabla *usuarios* y finalmente se seleccionaría toda la tabla *datos*, que no debería estar disponible para los usuarios comunes. En resumen, cualquier dato de la base de datos puede quedar disponible para ser leído o modificado por un usuario malintencionado.

Teniendo en cuenta lo anteriormente expresado es de vital importancia chequear y validar todos los elementos que intervienen en la ejecución de un script SQL en función de garantizar el correcto funcionamiento de nuestros sistemas, así como la satisfacción de los clientes para los cuales estos se desarrollaron. Por tanto GeneSIG provee un mecanismo de escape el cual se describe a continuación:

```
$query = "UPDATE !! SET nom_rolesid_rol=? WHERE id_usuario=?";
$result = $this->db->getAll(
    $query,
    array(
        $this->configSchema,
        'seg_perfilusuario',
        $idrol,
        $iduser
    )
);
```

Obsérvese que la función *getAll* recibe por parámetro la sintaxis SQL en modo de texto, indicándose con el operador ? cada vez que se requiera un valor variable y como segundo parámetro se le especifica el listado de valores requeridos en el mismo orden que se

especificaron con el operador anteriormente mencionado. Las secuencias de escape se usan dentro de una instrucción SQL para indicar al controlador que la parte de la secuencia de escape de la cadena SQL se debería tratar de forma diferente. Cuando el controlador PEAR procesa la parte de la secuencia de escape, traduce esa parte de la cadena en código SQL.

5.7.3 Extracción de información de las Layer

En función de facilitar el trabajo con las opciones propias de la mapscript el equipo de desarrollo se dio la tarea de desarrollar algunas utilidades, tal es el caso de *PostgisDataGenesig* definida en el directorio *common* en la raíz del marco de trabajo. Esta cuenta con un conjunto de funciones que permiten obtener información a partir de los datos contenidos en un objeto msLayer, las cuales se describen a continuación:

PostgisDataGenesig::obtenerDSNDeConexionPostgis es una función que permite obtener a partir de la propiedad *connection* de un objeto capa de la mapscript, la información referente a la fuente de conexión, para ello se le deben especificar dos parámetros los cuales se describen a continuación:

- **connection:** cadena de caracteres que contiene la información necesaria para extraer la información requerida.
- **tipoRetorno:** propiedad de tipo entera, cuyos rango de valores pueden ser [0,1], por defecto asume valor 0, indicando de esta forma que devolverá el resultado en forma de cadena con sintaxis *sdn*, y en caso contrario devolvería un arreglo asociativo a partir de índices semánticos [*user*, *pass*, *host*, *dbname*]

Este recurso es muy útil, por ejemplo en caso que se requiera obtener el objeto de conexión definido para una capa específica en función de realizar operaciones de lectura y escritura sobre los mismos, sería algo tan simple como:

```
$co = PostgisDataGenesig::obtenerDSNDeConexionPostgis($msLayerObj->connection);
$db = $this->getConnectionTo($co);
```

Es importante destacar que en no se pueden especificar dentro de las consultas in identificador de tipo alias, pues influye directamente en el mecanismo de paseado de la misma y se obtendrían resultados no deseados, a continuación se muestran un conjunto de sintaxis donde se demuestra dicha afirmación.

- incorrecto: select ff.a from nom_prueba.arbol ff
- correcto: select arbol.a from nom_prueba.arbol
- correcto: select a from nom_prueba.arbol
- correcto: select * from nom_prueba.arbol

PostgisDataGenesig::obtenerCamposConsulta como bien describe su nombre permite obtener los campos que componen una consulta, así como el campo clave, pues asume que las consultas de tipo postgis emplean un campo como filtro y de carácter único. En función de arrojar este tipo de resultados recibe dos parámetros los cuales se describen a continuación:

- **consulta:** cadena de caracteres que contienen la información necesaria en función de obtener el resultado esperado.
- **camposNoMostrar:** define el listado de campos que serán excluidos de la respuesta, toma como valor por defecto un arreglo vacío.

Como resultado arroja una estructura asociativa basada en índices semánticos tal y como se describe a continuación:

- llave: valor de tipo *string*
- campos: valor de tipo *array* cuyos índices son [*tipo, campo, alias, campoconsulta*]

Teniendo en cuenta estos elementos la sintaxis del mismo es la que se muestra en el ejemplo que se presenta a continuación:

```
$query = " SELECT * from elementos limit 1 ";
$db = $this->getConnectionTo("pgsql://$usuario:$password@$host/$baseDatos");
$campos = PostgisDataGenesig::obtenerCamposConsulta($query);
$campos = $campos['campos'];
```

PostgisDataGenesig::obtenerCuerpoConsulta permite obtener los elementos más relevantes definidos en el cuerpo de una consulta postgis. En función de arrojar el resultado esperado requiere que se le especifique el arreglo de caracteres que contienen la información necesaria, generalmente se refiere al campo data de una capa definida por la mapscript. El resultado lo arroja en forma de arreglo asociativo basado en índices semánticos [*from, using, geom, select, where*].

```
$query = $msLayerObj->data;
$infolay = PostgisDataGenesig::obtenerCuerpoConsulta($query);
$from = $infolay['from']
```

5.8 Mapscript desde dentro

Desde sus inicios CartoWeb fue concebido como un *framework* en función de recrear una capa de abstracción para el manejo de mapas con Mapserver a través de la librería Mapscript. De esta forma sería más fácil incorporar comportamientos que aun no estuviesen implementados por dicho servidor. Para lograr dicho objetivo implementaron la clase ServerContext la cual se encuentra ubicada en el directorio ./sever/ dentro de la raíz del propio marco de trabajo. Teniendo en cuenta las responsabilidades definidas por CartoWeb para cada uno de los elementos que componen un plugin, es posible acceder solamente desde la parte servidora a una instancia única de esta misma clase y de esta forma manejar los componentes intrínsecos de un mapa.

Es importante destacar que la filosofía de CartoWeb consiste en manejar perfiles cartográficos por usuario, ósea por cada cliente que esté utilizando la plataforma se genera un fichero de extensión .map a partir de una plantilla definida o del sistema que se encarga de la gestión de los mismos denominada LiberMap. Y sobre este se alancearía toda la interacción inducida por el mismo, a través de los recursos anteriormente mencionados. A continuación se muestra un ejemplo ilustra cómo obtener el objeto mapa:

```
$msMapObj = $this->serverContext->getMapObj();
```

Partiendo de lo expresado anteriormente en ocasiones se hace prioritario un mecanismo de almacenamiento de los cambios realizados en tiempo de ejecución sobre determinado mapa. Pese que la mapscript contiene una funcionalidad que de forma muy simple permite solventar esto, GeneSIG provee la función *ServerContext::saveMapObj*, permitiendo abstraer a los desarrolladores de identificar que usuario está conectado, en qué directorio y con qué identificador debe almacenarse su perfil cartográfico, tal y como se muestra a continuación:

```
$this->serverContext->saveMapObj();
```

Sin embargo esta no limita a que pueda almacenarse solo aquella información que haya

sido nidificada en tiempo de ejecución, pues este consta de un parámetro denominado *mapObj* cuyo valor por defecto se asume como *null* y describe un objeto de tipo *msMapObj*. Por ejemplo en caso que se desee sobrescribir los campos definidos en el mapa actual con uno creado completamente de forma dinámica sería algo muy similar a lo que se muestra a continuación:

```
$objMap = ms_newMapObj( "" );
$objMap->set( "name", "Cuba" );
$objMap->set( "status", MS_ON );

$objMap->setSize( 640, 480 );
$objMap->setExtent( -85, 17.7331838565022, -74, 26.2668161434978 );
$objMap->setProjection( "init=epsg:4267" );

$this->serverContext->saveMapObj( $objMap );
```

Esta clase además provee otras utilidades como son *getMaxExtent*, permitiendo obtener la máximo extensión de visualización para el mapa. Las funciones *resetMsErrors* y *checkMsErrors* permiten respectivamente limpiar la cache de errores detectados y realizar un cheque o exhaustivo de los *bug* detectados. Por otra parte las funciones *getMsVersion* y *isMsNewerOrSameThan* permiten conocer y establecer comparaciones referentes a la versión del servidor de mapas que esté utilizando el producto, así como a través de *getMapPath* es posible conocer el directorio en que se encuentra almacenado dicho perfil cartográfico.

ServerContext también está provista de otras funcionalidades que si bien no tiene que ver con la manipulación del mapa en sí mismo, si permite una mejor reutilización entre los componentes internos del marco de trabajo. Tal es el caso de la función denominada *getCorePluginNames* la cual permite obtener los nombres de los *coreplugin* que se encuentran activos. De igual forma el recurso *loadPlugins* permite iniciar la carga en memoria de las partes servidoras de los *plugin*, o *getProjectHandler* y *getPluginManager* implementada en función de acceder al controlador de proyecto y el manejador de *plugin*, para mayor comprensión de la utilización de este ultimo refiérase al acápite 5.5.4 *Comunicación entre plugins*.

5.9 Recursos

La plataforma GeneSIG posee una serie de recursos que implementan una gran cantidad de comportamientos que son generalizados y con una marcada tendencia a influir en los restantes módulos de forma horizontal, por tanto siguiendo con la política de reducir esfuerzos se describen a continuación que peculiaridades estos poseen y como se pueden utilizar.

5.9.1 Base

En GeneSIG se definió un *coreplugin* con la responsabilidad de conformar el soporte para toda la infraestructura de la parte cliente, la cual consta desde el API para la manipulación del mapa en JavaScript, hasta los mecanismos definidos para gestionar la interfaz gráfica de usuario. La utilidad principal de este modulo es que en aquellos proyectos que requieran utilizar otra librería como OpenLayers para el tratamiento de mapas y geometrías sobre estos, u otra librería GUI como JQuery por tan solo citar un ejemplo.

En esta versión el *coreplugin* provee una capa de abstracción para la manipulación de interfaces gráficas sobre GeneSIG, basadas en la estructura organizativa que propone el componente Viewport de ExtJs. Esta fue denominada GView y se encuentra ubicada en

`./coreplugins/base/htdocs/js/jsClass/` a partir de la raíz del propio marco de trabajo. Presenta un conjunto de funcionalidades que permiten de forma transparente para los desarrolladores adicionar elementos de GUI al sistema sin que estos impliquen grandes modificaciones.

A continuación se procede a describir los principales elementos que componen la GUI propuesta para la plataforma GeneSIG:

- **A:** Define el área dedicada al título de la aplicación, así como su logo.
- **B:** Espacio reservado para el menú contextual, al igual que la opción A forma parte de la región norte.
- **C:** Área definida como barra de herramientas, espacio donde se ponen los principales elementos de interacción entre el usuario y el sistema, los cuales evidentemente aparecen en el menú contextual.
- **D:** Región central del sistema, dedicado exclusivamente a la visualización y trabajo sobre el mapa.
- **E:** Región este, define un espacio para incorporar herramientas que tengan una incidencia de carácter horizontal sobre el resto, así como una influencia directa sobre el mapa, generalmente queda reservado para el controlador de capas.
- **F:** Región oeste, es un área para anclar aquellas herramientas que requieran un cumulo considerable de opciones, al punto que complejicen el trabajo con el mapa teniendo en cuenta que limiten su visualización en caso de utilizar otros recursos como ventanas flotantes.
- **G:** Región sur, es considerado un pequeño espacio para visualizar contenido relevante relacionado con el mapa de carácter informativo, tal es el caso del par de coordenadas, la escala, el factor de Zoom, etc.

Para mayor comprensión véase la figura 12, donde se visualizan cada uno de los elementos descritos anteriormente.

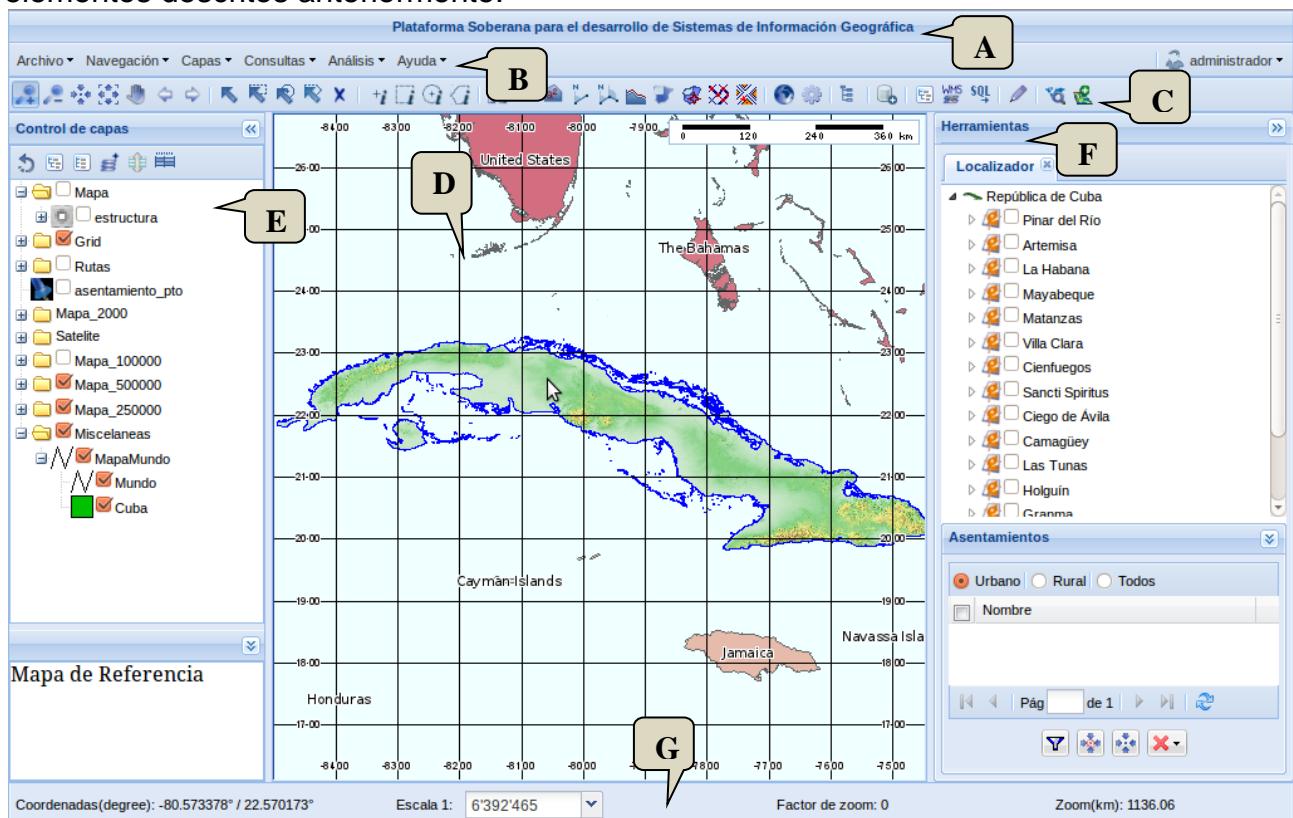


Ilustración 32 Vista general de la GUI para GeneSIG

En caso que se requiera adicionar un botón a la barra de herramientas que se encuentre en la parte superior del sistemas se podría utilizar la función `GView::addToolBarButton`, esta recibe por parámetro un objeto con las características necesarias para asegurar el correcto funcionamiento del mismo, entre estas las más significativas son:

- **handler**: en la cual se define el comportamiento del botón.
- **id**: define el identificador del botón.
- **ttip**: contiene el texto que se muestra una vez que pase el mouse por en cima del botón.
- **icon**: define el identificador de la clase que contiene la imagen o iconografía del botón.
- **text**: cadena de caracteres que será visualizada dentro del cuerpo del botón.

Teniendo en cuenta estos elementos si se desea adicionar un botón en la barra de herramientas denominado *myaccion*, quedaría de la siguiente forma:

```
objView.addToolBarButton({  
    text: "myaccion",  
    icon:"myaccion_cls",  
    ttip:"myaccion to do",  
    id: 'myaccion_id',  
    handler: function(){ ... }  
});
```

Obsérvese que se ha definido como variable global *objView* la cual resulta ser una instancia de la clase *GView*, permitiendo evidentemente acceder a todas la funcionalidades que esta provee. Por otra parte también es posible adicionar un elemento al menú contextual definido en la plataforma y es posible a través de la función `GView::addMenuBarButton`, especificándose por parámetro el objeto que describe ese nuevo ítem, tal y como se muestra a continuación:

```
objView.addMenuBarButton({  
    text: "myaccion",  
    icon:"myaccion_cls",  
    id: 'myaccion_id',  
    handler: function(){ ... }  
});
```

También es posible adicionar separadores en aras de establecer agrupaciones de herramientas, para ello se utiliza la función `GView::addSeparator`, tal y como se muestra a continuación:

```
objView.addSeparator();
```

Y en caso que se desee adicionar como submenú de algún ítem existente se deberá utilizar `GView:: addMenuBarButtonTo`, al cual se le especifica el ítem y el padre del mismo, a continuación se muestra un ejemplo de cómo se podría utilizar:

```
objView.addMenuBarButtonTo ({  
    text: "myaccion-child",  
    icon:"myaccion-child_cls",  
    id: 'myaccion-child_id',  
    handler: function(){ ... }  
}, 'myaccion_id');
```

Pese a las ventajas de las funcionalidades anteriormente mencionadas la más utilizada en este sentido es `GView::addMenuBarAction`, pues tan solo se le especifica el manipulador de la acción el cual puede ser una función anónima como se muestra en el ejemplo que

se ilustra a continuación y el identificador que contiene una restricción de nomenclatura, la cual consiste en especificar como prefijo el término *menu_* al identificador del *plugin* que coincide con el del botón previamente adicionado a la barra de herramientas.

```
objView.addMenuBarAction('menu_layerQuery', function(){ ... });
```

Otro elemento importante es la capacidad de adicionar paneles a las diferentes regiones definidas en el componente rector de la interfaz gráfica para la plataforma. Por ejemplo en caso de requerir adicionar un panel en el área inferior, en función de visualizar alguna información puntual como es el caso de la escala, se deberá utilizar la función *GView::addComponents*, tal y como se muestra a continuación:

```
objView.addComponents(new GView.component({
    properties: {
        layout:'column',
        xtype: 'panel',
        items: [
            {
                columnWidth: 0.05,
                style:'margin:5 0 0 0;',
                html: 'Escala 1'
            },
            {
                columnWidth: 0.45,
                style:'margin:0 0 0 5;',
                items:AjaxPlugins.Location.comboEscala
            }
        ]
    }, 'sur')
});
```

Otra de las opciones comúnmente utilizadas es la de adicionar en la región derecha un *tabpanel*, principalmente para aquellos *plugin* que requieran interfaces con cierto nivel de interactividad sobre el mapa, como es el caso de la edición o la exportación a *pdf* por citar algunos ejemplos. De esta forma se evita la incomodidad de estar moviendo constantemente una ventana flotante. Para lograr este objetivo también se utiliza la función *GView::addComponents*, a continuación se muestra un fragmento del *plugin* de exportación a *pdf* el cual hace uso de este recurso.

```
objView.addComponents(new GView.component({
    title: "Exportar Mapa",
    id: 'tab-export-pdf',
    region: 'centro',
    properties: {
        frame: true,
        closable: true,
        items: [ tabPanel ],
        buttons:[{
            text: 'Exportar',
            value: 'Print',
            handler: function() {...}
        },{
            text: 'Restaurar',
            handler: function() {...}
        }]
    }
}), 'este');
```

Otro elemento importante a destacar es el hecho de controlar las herramientas que se encuentran activas y las que no, pues de ello depende el comportamiento del sistema en sentido general. En función de facilitar el trabajo en este sentido, se definió un objeto global que provee algunas funcionalidades en este sentido, el cual se denomina *Genesig.Tools*. A continuación se muestra un ejemplo relacionado con el anterior, que ilustra cómo una vez desactivada la herramienta de exportación a *pdf*, se devuelve el

control a la herramienta que asume por defecto el mapa, en este caso seria del modulo navegación el zoom.

```
Ext.getCmp('tab-export-pdf').on('close', function() {
    Genesig.Tools.enableDefaultTool();
});
```

Obsérvese que utilizando la función *Ext.getCmp* se accede al objeto denominado *tab-export-pdf*, asignándosele una función anónima en el evento *onClose*, en la cual se emplea la función *Genesig.Tools.enableDefaultTool* permitiendo de esta forma lograrse el objetivo propuesto.

5.9.2 Layer

El *coreplugin* responsable de la gestión relacionada con la información que se visualiza o no sobre un mapa, así como las agrupaciones en las que estas se organizan se denomina Layer. En esta versión de GeneSIG el componente GUI rector de dicha tarea es un TreePanel definido en la librería ExtJs. Entre las utilidades implementadas por el mismo, en función de facilitar el trabajo a los desarrolladores se encuentra: *AjaxPlugins.Layers.addItemMenu* permitiendo crear de forma dinámica un menú contextual que se visualiza con la acción del clic derecho del mouse. A continuación se muestra un ejemplo de cómo un *plugin* puede adicionar un comportamiento de forma dinámica *coreplugin* al controlador de capas, en este ejemplo que se muestra a continuación sería un fragmento del layerQuery:

```
AjaxPlugins.Layers.addItemMenu({
    text: "Construir consulta",
    icono:"layerquery",
    handler: function(menu)
    {
        if( AjaxPlugins.LayerQuery.layerQ != undefined ){
            Genesig.Tools.enableTool( 'layerQuery', 1 );
            AjaxPlugins.LayerQuery.layerQ.cargarDatos( menu.node.attributes.text );
        } else loadDynamicConfig.executeLoadPlugin("layerQuery", function(){
            Genesig.Tools.enableTool('layerQuery',1);
            AjaxPlugins.LayerQuery.layerQ.cargarDatos(menu.node.attributes.text);
        });
    }
});
```

A esta función se le especifican tres atributos fundamentales, los cuales se describen a continuación:

- **text**: cadena de texto que se visualizan entre las opciones del menú.
- **icono**: identificador de la clase que describe como se visualiza el contenido anterior.
- **handler**: función manejadora del evento onclic sobre la acción previamente adicionada

Para mayor comprensión véase la figura que se muestra a continuación:

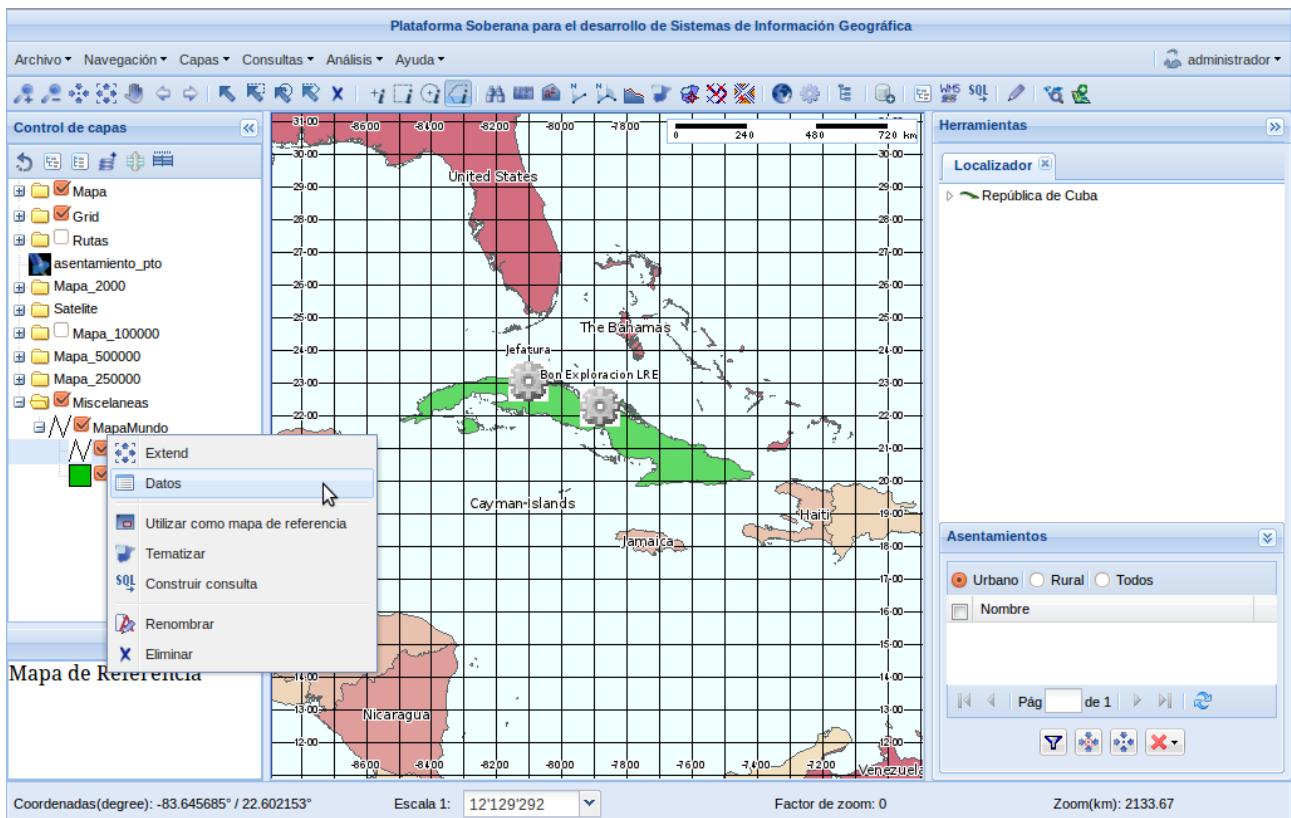


Ilustración 33 Vista del menu contextual sobre el controlador de capas

5.9.3 Location

EL *coreplugin* que engloba el comportamiento asociado al control de la navegación se denomina *Location*, el cual consta de un cumulo de elementos que pueden ser modificados a través de su propio fichero de configuración, a continuación se procede a describir los mismas:

Configuración definida en el modulo cliente del *coreplugin*:

- **scalesActive:** Propiedad de tipo boolean tomando valores comprendidos en [true/false], toma valor falso por defecto, en caso de que sea positiva se visualizan en un listado las escalas activas.
- **recenterActive:** Propiedad de tipo boolean tomando valores comprendidos en [true/false], toma valor falso por defecto, en caso de ser positiva se visualiza un formulario con las coordenadas del punto para recentro.
- **idRecenterActive:** Propiedad de tipo boolean tomando valores comprendidos en [true/false], toma valor falso por defecto, en caso de ser positivo seria visualizado el identificador de recentrado.
- **shortcutsActive:** Propiedad de tipo boolean tomando valores comprendidos en [true/false], toma valor falso por defecto, define si se visualiza el listado de accesos directos que simula el comportamiento de un localizador territorial.
- **scaleUnitLimit:** Define el limite a tener en cuenta el valor de la escala una vez expresado en alguna unidad de medida, tomando como base *m*.
- **panRatio:** Define el radio que abarca la zona geográfica afectada por la acción *pan*, la cual toma valor 1 por defecto, en caso de tomar valor por debajo de 1 podria implicar superposicionamiento o solapamiento de algunas zonas entre el mapa actual y el resultante de dicha acción y en caso contrario en el que

tomaría valor inferior a 1, implicaría saltos una vez comparadas las dos vistas.

- **weightZoomIn:** Propiedad numérica que define el orden de visibilidad para el comportamiento *ZoomIn* sobre la barra de herramienta, toma valor 10 por defecto y en caso de que se especifique un valor negativo desabilita dicha herramienta.
- **weightZoomOut:** Propiedad numérica que define el orden de visibilidad para el comportamiento *ZoomOut* sobre la barra de herramienta, toma valor 11 por defecto.
- **weightPan:** Propiedad numérica que define el orden de visibilidad para el comportamiento *Pan* sobre la barra de herramienta, toma valor 12 por defecto.
- **crosshairSymbol:** Propiedad entera que define a través de un valor numérico el identificador del símbolo utilizado para inducir el punto de recentrado del mapa.
- **crosshairSymbolSize:** Propiedad numérica que define el tamaño de visualización para el símbolo utilizado en el recentrado del mapa.
- **crosshairSymbolColor:** Define el valor del color expresado en el sistema RGB, donde los números deberán ser especificados concatenados a través del carácter “,”.
- **showRefMarks:** Propiedad de tipo boolean tomando valores comprendidos en [true/false], en caso positivo permitirá visualizar las marcas de referencias, las cuales son configuradas en el servidor.

Configuración definida en el modulo servidor del *coreplugin*:

- **minScale:** En caso de ser especificada define el minimo valor que puede tomar la escala.
- **maxScale:** En caso de ser especificada define el máximo valor que puede tomar la escala.
- **scaleModeDiscrete:** Propiedad de tipo *boolean* tomando valores comprendidos en [true/false], en caso positivo la escala debe ser especificada de forma manual.
- **zoomFactor:** Se define como el valor en que aumenta o disminuye una acción de zoom, se utiliza siempre y cuando la propiedad *scaleModeDiscrete* ala sido inicializada en false.
- **noBboxAdjusting:** Propiedad de tipo *boolean* tomando valores [true/false], toma por defecto valor false. En caso de ser inicializada en true todos los datos comprendidos en el *extent* definidos por el *.map* serán visualizados en la acción de *fullextend*, siendo este el comportamiento por defecto que asume Mapserver, en caso contrario no será visualizado lo que se encuentre fuera del *extend* inicial.
- **scales:** Se define como un listado de objetos que describen los valores de la escala en caso de ser especificada en *true* la propiedad *scaleModeDiscrete*. Las claves que pueden ser especificadas son [*label*, *value*, *visible*], las cuales hacen referencia respectivamente al texto que describe la escala, el valor numérico que toma la misma, así como si se desea o no que se visualice, a continuación se muestra un ejemplo que ilustra dicho comportamiento:

```
scales.0.label = 1/250 000  
scales.0.value = 250000  
scales.0.visible = true
```

```
scales.1.label = 1/500 000  
scales.1.value = 500000  
scales.1.visible = true
```

```

scales.2.label = 1/750 000
scales.2.value = 750000
scales.2.visible = true

scales.3.label = 1/1 500 000
scales.3.value = 1500000
scales.3.visible = true

```

- **shortcuts:** Esta propiedad es también conocida en otros ambientes como *acceso rápido*, contiene un listado de objetos que describen los elementos necesarios en función de brindar un acceso rápido a la información, en otras palabras este recurso simula el comportamiento de un localizador territorial, las claves que pueden ser especificadas son [*label*, *bbox*], las cuales hacen referencia respectivamente al identificador semántico que describe el acceso, así como la caja que delimita dicha área geográfica, también conocida como *extend*, lo que en este caso los valores estarían concatenados con el carácter “,”, a continuación se muestra un empleo de su empleo:

```

shortcuts.0.label = Mont Blanc
shortcuts.0.bbox = "280000, 5050000, 380000, 5100000"

shortcuts.1.label = Leman Lake
shortcuts.1.bbox = "260000, 5110000, 360000, 5180000"

shortcuts.2.label = Albertville
shortcuts.2.bbox = "225000, 5000000, 370000, 5110000"

```

Un elemento clave a destacar es que la sintaxis de la nomenclatura definida para la propiedad denominada como *bbox* es: *xmin,ymin,xmax,ymax*

- **recenterMargin:** Propiedad numérica, que define el margen adicionado alrededor del centro para el mapa, expresa el porcentaje que representaría la combinación [*width/height*], en caso de no ser especificada se pondría en práctica el mecanismo definido por la propiedad *recenterDefaultScale*.
- **recenterDefaultScale:** Propiedad de valor numérico, permite ajustar la escala una vez que se halla recentrado el mapa, principalmente cuanto dicha acción se basa en un punto.
- **refMarksSymbol:** Símbolo utilizado para dibujar líneas en función de crear marcas de referencia, típicamente se emplean elipses.
- **refMarksSymbolSize:** Tamaño expresado en pixel, en función de la propiedad *refMarksSymbolSize*, usualmente se emplean valores como [1,2,...].
- **refMarksSize:** Define el tamaño para las marcas de referencia, una vez especificado no varia en función de la escala.
- **refMarksColor:** Describe el valor del color para las marcas, expresado en el sistema RGB, donde los valores deberán ser concatenados por el carácter “,”
- **refMarksTransparency:** Atributo numérico que define el grado de transparencia de las marcas.
- **refMarksOrigin:** Describe el punto de origen expresado en el sistema (X,Y), donde los valores deberán ser concatenados con el carácter “,”, típicamente toma valores “0, 0”.
- **refMarksInterval:** Se define como un contenedor de objetos que describen los diferentes intervalos de marcas, las claves que se pueden especificar son [*maxScale*, *interval*], las cuales caracterizan la máxima escala correspondiente para dicho intervalo, así como el valor real del intervalo entre marcas, separados por los valores definidos en el sistema (X,Y), a continuación se

muestra un ejemplo que ilustra su empleo:

```
refMarksInterval.0.timeScale = 200000
refMarksInterval.0.interval = "10000, 10000"

refMarksInterval.1.timeScale = 750000
refMarksInterval.1.interval = "50000, 50000"

refMarksInterval.2.timeScale = 2000000
refMarksInterval.2.interval = "100000, 100000"

refMarksInterval.3.timeScale = 5000000
refMarksInterval.3.interval = "250000, 250000"
```

- **refLinesActive**: Propiedad de tipo *boolean* tomando valores comprendidos en [true/false], en caso de tomar valor positivo visualizara las marcas de referencia en los bordes de la pagina.
- **refLinesSize**: Define el tamaño que tomara las marcas una vez situada sobre los límites de la pagina.
- **refLinesFontSize**: Define el tamaño de la fuente de letra que contendrá aquellas marcas de referencias situada sobre los límites de la pagina.

Un elemento a destacar es que la mayoría de los elementos configurables definidos en el modulo cliente principalmente están en función de la antigua API basada en DHTML, las cuales son prácticamente obsoletas a partir de la incorporación de ExtJs. No obstante hay muchis recursos que ya están implementados y en un futuro no muy lejano se le debe incorporar soporte a la nueva capa de abstracción para GUI.

Independientemente de los elementos de configuración definidos por este *coreplugin*, también provee un cumulo considerable de funcionalidades implementadas principalmente por la parte servidora que garantizan evitar la duplicidad de esfuerzos, y de esta forma reutilizar dichos comportamientos.

ServerLocation::doLocation es la función que permite llevar a cabo un desplazamiento sobre el mapa, para ello requiere que se le especifiquen dos parámetros: *bbox* y *scale*, los cuales se describen a continuación:

- **bbox**: instancia de la clase *Bbox* definida por GeneSIG, la cual describe un rectángulo conformado por dos puntos que delimitan la extensión territorial que será visualizada en el mapa. Esta clase requiere de cuatro parámetros los cuales se definen en el orden que se muestran a continuación: *xmin,ymin,xmax,ymax*
- **scale**: Propiedad numérica que define el valor de la escala definida para dicha localización.

A continuación se muestra un ejemplo de cómo modificar el area de visualización en el mapa y dirigirla hacia un destino específico:

```
$box = array(
    "minx" => -83.493143,
    "miny" => 22.240536,
    "maxx" => -75.922997,
    "maxy" => 20.050964
);
$scale = 7675459;
$bbox = new Bbox( $box["minx"], $box["miny"], $box["maxx"], $box["maxy"] );
$pluginmanager = $this->serverContext->getPluginManager();
$pluginmanager->location->doLocation($bbox, $scale);
```

Obsérvese que se utiliza el recurso *ServerContext::getPluginManager* en función de

acceder al *coreplugin*.

Tambien se encuentra provisto por métodos de carácter informativo, tal es el caso de: *getScales*, *getScaleFromBbox* y *getLocationResult*, las cuales respectivamente muestran un listado de escalas soportadas por el mapa en ejecución, retornan el valor la escala en función de un objeto Bbox, así como un objeto de localización resultando el cual contienen la extencion territorial que se esta visualizando en un instante de tiempo determinado, tal y como se muestra a continuación:

```
$pluginManager = $this->serverContext->getPluginManager();
$listOfScalesMap = $pluginManager->location->getScales();
$scaleFromBox = $pluginManager->location->getScaleFromBbox($bbox);
$objLocationRes = $pluginManager->location->getLocationResult();
```

Tambien presenta metodos de localización territorial, tales como *doLocation* la cual fue brevemente descrita anteriormente, también se encuentra la función *doLayerFullExtent*, esta ultima permite mostrar en pantalla el area geográfica donde se muestra a plenitud todo el contenido cartográfico que componen determinada capa en el mapa, en función de lograr esto es necesario especificar los parámetros que se detalla a continuación:

- **layerName:** Define el nombre que identifica a capa
- **filtro:** Listado de condiciones que permiten filtrar la información contenida en dicha capa

A continuación se muestra pequeño fragmento de código del propio *coreplugin Location* que ilustra como puede ser utilizado esta funcionalidad:

```
$request->capaExtent = $request->capaExtent != "" ? "asentamiento_pto" : $request->capaExtent;
$pluginManager = $this->serverContext->getPluginManager();
if($request->capaExtent) {
    $this->doLayerFullExtent( $request->capaExtent, $request->capaFiltro);
    return ;
}
```

5.9.4 Query

Query es la denominación que se le concede al *coreplugin* responsabilizado de búsqueda de objetos geográficos que se encuentren en un área determinada sobre el mapa activo. Un elemento clave en los procesos de identificación y selección de geometrías sobre el mapa activo, es la especificación de que capas serán tenidas en cuenta en los mismos, partiendo de que estos procesos se realizan a nivel de clases. Es por ello que la herramienta visual de GeneSIG muestra una ventana donde obliga al menos especificar una capa, para mayor comprensión véase la ilustración 12.

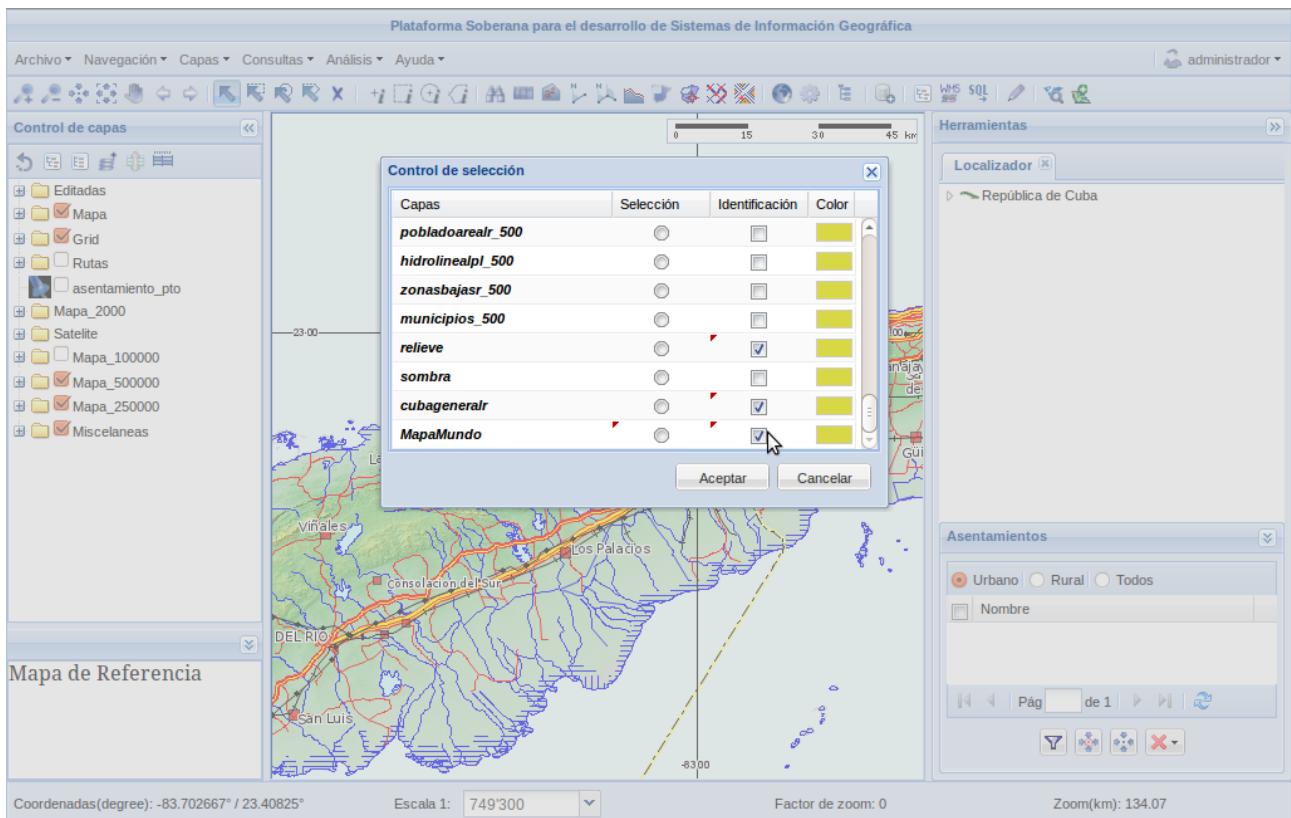


Ilustración 34 Interfaz para la gestión de capas seleccionables y/o consultables

Alguno de los elementos mas relevantes que componen la parte cliente de este coreplugin, se describen a continuación:

- **persistentQueries:** Propiedad de tipo *boolean* tomando valores comprendidos en [*true/false*], toma valor falso por defecto. En caso de ser especificada en verdadero la información consultada se mantendría de forma persistente y en caso contrario se perderían los datos una vez recargada la aplicación.
- **id_attribute_string:** Esta propiedad describe el identificador del atributo que deberá ser utilizado como *id* en el procesos de identificación y se basa en las especificaciones que se resobre el metadato de determinada capa en el fichero de extensión *.map*, a continuación se muestra un ejemplo:

```

METADATA
    "id_attribute_string" "FID | string"
    "query_returned_attributes" "FID FNAME"
END
  
```

Observese que se especifica valor el par denominado [*name/type*], los cuales hacen referencias al identificador semántico de la propiedad y al tipo de dato que la describe.

- **returnAttributesActive:** Propiedad de tipo *boolean* tomando valores comprendidos en [*true/false*], toma valor falso por defecto. En caso de ser especificada de forma positiva una vez realizada la consulta se obtendrán todos los atributos asociados a dichas geometrías en caso contrario se obtendrían tan solo los id.
- **defaultPolicy:** define el tipo de política o mecanismo de selección tomando valores como:
 - POLICY_XOR: exclusión
 - POLICY_UNION: unión
 - POLICY_INTERSECTION: intersección
 - POLICY_REPLACE: remplazo

- **defaultHilight:** Propiedad de tipo *boolean* tomando valores comprendidos en [true/false], toma valor verdadero por defecto. En caso de ser positiva una vez realizada la consulta se procederá a resaltar las geometrías identificadas.

Independientemente de las opciones de configuración que el modulo presente, también es posible utilizarlo desde otros plugin en aras de ganar por concepto de reutilización. Tal es el caso de la función *ServerQuery::queryByShape*, la cual permite extraer todas aquellas geometrías contenidas en un area determinada, para ello es necesario que se le especifique dos parámetros:

- **layerId:** Identificador semántico de la capa
- **shape:** Objeto de tipo *Shape*, el cual contiene la información necesaria para efectuar dicha consulta.

A continuación se expone un fragmento del código contenido en la parte servidora del plugin *identificación*, en el cual se pone de manifiesto el empleo del *Query*:

```
$results = array();
$pluginManager = $this->serverContext->getPluginManager();
foreach( $request->layersArray as $layer) {
    $msLayer = $this->getLayerByName($layer);
    $result = $pluginManager->query->queryByShape($layer, $request->shape);
    $results[] = $result;
    $pluginManager->query->highlightQuery($result, $layer);
}
```

Observese que en este ejemplo se conoce el listado de capas que serán consultadas a través de la propiedad *layersArray* del objeto *request*, así como el objeto de tipo *shape*, el cual define el tipo de consulta a efectuar: puntual, rectangular, radial o poligonal. Este objeto generalmente se conforma en la parte cliente del *plugin* en que se desee llamar al *Query*, y se puede generar de forma manual a través de una petición por *POST/GET*, o utilizando un recurso propio de la plataforma que se integra con la API cliente, en función de abstraer a los desarrolladores de estos quehaceres, a continuación se muestra un ejemplo que ilustra con mas claridad lo panteado.

```
$queryRequest = new QueryRequest();
$queryRequest->layersArray = array("provincias_r", "asentamientos", "costas");
$queryRequest->shape = $this->cartoclient->getCartoForm()->mainmapShape;
```

Como bien se puede observar la función *Cartoclient::getCartoForm*, permite obtener un objeto de tipo *CartoForm* cual contiene en la propiedad *mainmapShape* una instancia de la clase *Shape*, generada desde la propia API cliente. Esta ultima en función de la herramienta generada puede tomar valores como los que se muestran a continuación:

- Herramienta de selección puntual

```
CartoForm Object
(
    [pushedButton] => 2
    [mainmapShape] => Point Object
    (
        [x] => -83.46286599039932
        [y] => 22.58553231953649
        [className] => Point
    )
    [keymapShape] =>
)
```

- Herramienta de selección rectangular

```
CartoForm Object
(
    [pushedButton] => 2
```

```

[mainmapShape] => Rectangle Object
(
    [minx] => -83.58870235896862
    [miny] => 22.58553231953649
    [maxx] => -83.47948588813489
    [maxy] => 22.690004449902805
    [className] => Rectangle
)
)
[keymapShape] =>
)

```

- Herramienta de selección radial

```

CartoForm Object
(
    [pushedButton] => 2
    [mainmapShape] => Circle Object
(
    [x] => -83.58870235896862
    [y] => 22.773107280876008
    [radius] => 0.206165494529003
    [className] => Circle
)
)
[keymapShape] =>
)

```

- Herramienta de selección poligonal

```

CartoForm Object
(
    [pushedButton] => 2
    [mainmapShape] => Polygon Object
(
    [points] => Array
(
    [0] => Point Object
(
    [x] => -83.64805913659565
    [y] => 22.60215288573113
    [className] => Point
)
    [1] => Point Object
(
    [x] => -83.66230476322613
    [y] => 22.462065256376302
    [className] => Point
)
    [2] => Point Object
(
    [x] => -83.55783683460257
    [y] => 22.443070323582425
    [className] => Point
)
    [3] => Point Object
(
    [x] => -83.44387182155867
    [y] => 22.473937089372473
    [className] => Point
)
    [4] => Point Object
(
    [x] => -83.52934558134159
    [y] => 22.65438895091429
    [className] => Point
)
    [5] => Point Object
(
    [x] => -83.64805913659565
    [y] => 22.60215288573113
    [className] => Point
)
)

```

```

        )
        [className] => Polygon
    )
    [keymapShape] =>
)

```

Por otra parte el resultado arrojado por la función `ServerQuery::queryByShape` está compuesto por objetos de tipo `ms_shape_obj`, con una sintaxis similar a la que se muestra a continuación:

```

Array
(
    [0] => ms_shape_obj Object
    (
        [_handle_] => Resource id #480
        [numlines] => 421
        [type] => 2
        [index] => 59
        [tileindex] => 0
        [classindex] => 0
        [numvalues] => 22
        [text] =>
        [bounds] => ms_rect_obj Object
        (
            [_handle_] => Resource id #481
            [minx] => -84.957428
            [miny] => 19.828083
            [maxx] => -74.131775
            [maxy] => 23.594925
        )
        [values] => Array
        (
            [gid] => 59
            [name] => Cuba
            [iso_3166_2] => CU
            [iso_3166_3] => CUB
            [iso_3166_n] => 192
            [internet] => CU
            [fips_10_4] => CU
            [com] =>
            [pop_2003] => 11263429
            [pop_1990] => 10544798
            [pop_2010] => 11526447
            [births] => 11.87
            [deaths] => 7.38
            [migration] => -1.05
            [p_natgrowth] => 0.45
            [p_growthrate] => 0.34
            [inf_mortb] => 7.15
            [inf_mortm] => 8.06
            [inf_mortf] => 6.19
            [lifeexp_b] => 76.80
            [life_expm] => 74.38
            [life_expf] => 79.36
        )
    )
)

```

Evidentemente esta información depende de la estructura o taxonomía de la capa que se ha utilizado para realizar dicha consulta. Otro elemento muy utilizado y que también se mostró en el ejemplo anterior es la posibilidad de resaltar con un estilo definido aquellas geometrías encontradas, esto es posible a través de la función `ServerQuery::highlightQuery`, la cual recibe por parámetro la geometría y el nombre de la capa a la que esta pertenece, para mayor comprensión véase la figura 13, donde se ilustra la puesta en práctica de lo planteado anteriormente.

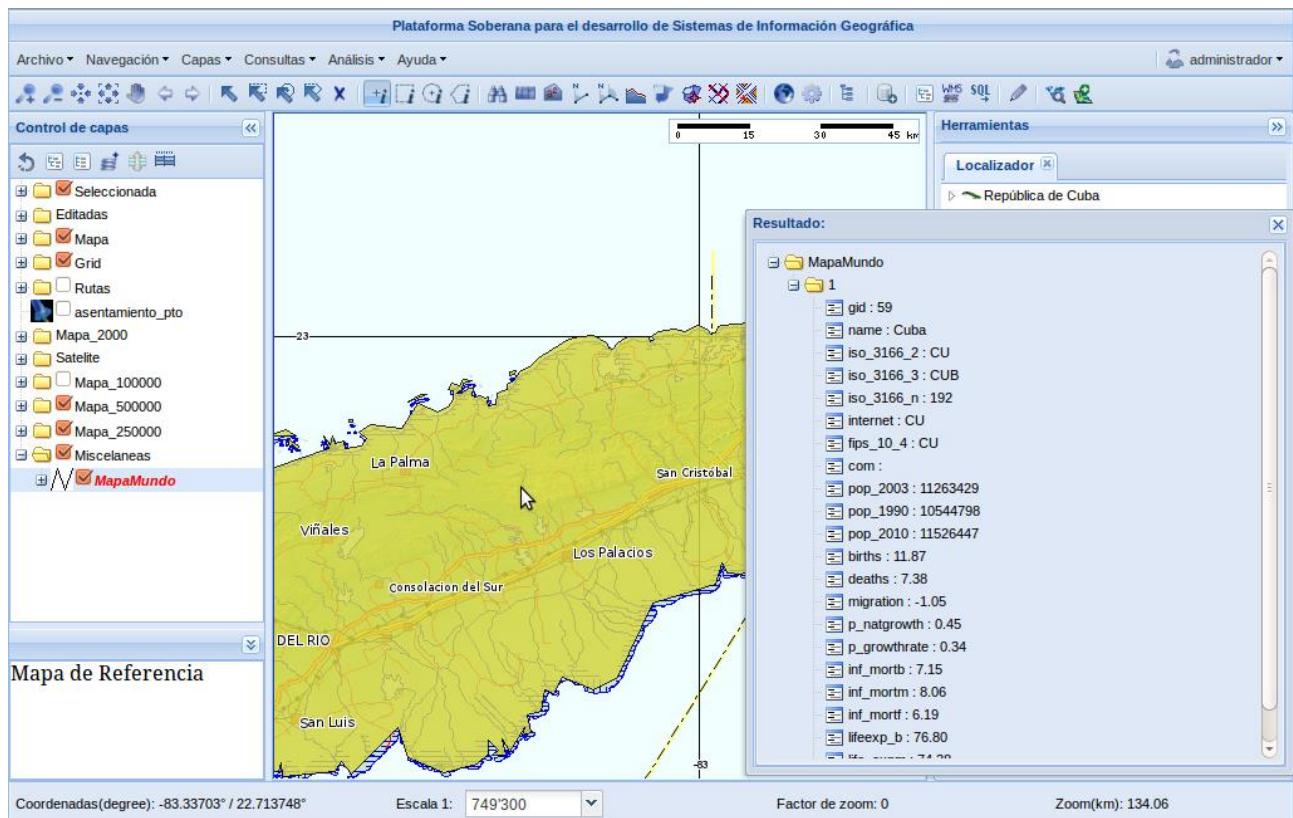


Ilustración 35 Vista que represa la acción de consulta o identificación sobre el mapa

5.9.5 Auth

El plugin encargado de proveer la garantía que asegura la certeza o autenticidad de que los usuarios que acceden al sistema, sean aquellos cuyos privilegios se les han sido asignado para este fin. En función de lograr el objetivo planteado se emplea una clase denominada *SecurityManager*, ubicada a partir del directorio raíz en *./common/SecurityManager.php*, la cual se encarga del proceso de factorización de aquellas especializaciones del contenedor de seguridad, así como la gestión de sus procesos internos. Este último fue denominado *SecurityContainer*, y define un conjunto de funciones abstractas que deben ser implementadas en sus predecesoras.



Ilustración 36 Vista del modulo para la autenticación de la plataforma

SecurityContainer::checkUser esta función permite chequear si un usuario determinado

es válido y a introducido correctamente su clave de acceso, por tanto debe retornar un valor comprendido en [true/false], para ello requiere que se especifiquen por parámetro el identificador del usuario, así como su contraseña.

```
class ModConfigSecurityContainer extends SecurityContainer
{
    public function checkUser( $username, $password )
    {
        $password = md5( $password );
        if( $dbpassword == $password && $dbusername == $username )
            return true;
        else return false;
    }
}
```

SecurityContainer::getRoles esta funcionalidad devuelve un listado de roles asociados al identificador de un usuario determinado, que se le debe especificar por parámetro, a continuación se muestra un fragmento de código que ilustra una posible implementación.

```
require_once(CARTOWEB_HOME . 'include/wrapperpear/DB.php');
require_once(CARTOWEB_HOME . 'client/Menu.php');
class ModConfigSecurityContainer extends SecurityContainer
{
    public function getRoles( $username )
    {
        $db = DB::connect($this->config);
        if( $DB instanceof DB_Error )
            throw new CartocommonException('Error de conexi&on a la DB');
        $query = "SELECT rules FROM mod_configuracion.seg_perfilusuario
                  where usuario='$username'";
        $db->setFetchMode( DB_FETCHMODE_ASSOC );
        $result = $db->getAll( $query );
        return $result;
    }
}
```

SecurityContainer::getTools esta función retorna el listado de herramientas a las que tiene acceso un usuario determinado, estas deben estar concatenadas en una única cadena a través del carácter “,”. A continuación se muestra un fragmento que ilustra una posible implementación.

```
class seguridadsoap
{
    public function seguridad()
    {
        return 'alsocronico,analisisTematicoCipd,btt_ATisoCap,
                btt_RadialBus,btt_editXYCaptura,btt_isoCaptura,
                btteleminarCIPD,busquedaRadialCipd,localizadorCipd,
                reportCipd,configuracion,altaBajaCipd,control_seleccion';
    }
}

class SoapAcaxiaSecurityContainer extends SecurityContainer
{
    public function getTools()
    {
        $ssoap = new seguridadsoap("", "");
        return $ssoap->seguridad();
    }
}
```

Como bien se había explicado anteriormente cada una de estas especializaciones deben ser ubicadas en el directorio `./plugins/auth/common/` y debe especificarse en el fichero de configuración para el plugin auth denominado `./client_conf/auth.ini`, específicamente la propiedad `securityContainer`, a continuación se muestra un ejemplo de cómo podría ser

definido:

```
securityContainer= SoapAcaxia
dbConfig = "pgsql://ing:ingsig09Grm@10.12.170.140/Genesig"

urlSeguridadUser=http://10.12.170.159:3128/seguridad/webservices/AcaxiaSoapAutentication.wsdl
urlSeguridadFunction=http://10.12.170.159:3128/seguridad/webservices/SeguridadSoapService.wsdl
identidadestructura=100000001
idsistema=300000000034

[views]
    compresJS = 0
    staticLoad[] = auth.ajax.js
    cssLoad[] = auth.css
[Menu]
    login.sessiondown.id = "menu_auth"
    login.sessiondown.name = "autenticacion"
    login.sessiondown.text = "Cerrar sesión"
    login.sessiondown.iconCls = ""
    login.sessiondown.type = "tool"
```

Notese que a la propiedad `securityContainer` tan solo se le asigna el valor de `SopAcaxia` contrario de `SopAcaxiaSecurityContainer`, teniendo en cuenta que el prefijo de `SecurityContainer`, se gestiona internamente en el marco de trabajo.

En caso que se requiera redefinir el comportamiento de este modulo o simplemente desarrollarlo completamente, se debe utilizar la clase `SecurityManager`, a continuación se muestra un ejemplo que ilustra como se podría implementar el comportamiento de `login`:

```
public function login( $usuario , $password )
{
    $config = $this->config;
    $manager = SecurityManager::getInstance();
    $className = ucfirst( $config->securityContainer ) . 'SecurityContainer';
    $manager->setSecurityContainer( new $className( $config ) );
    $login = $manager->checkUser( $usuario, $password );
}
```

Uno de los elementos claves descritos en el ejemplo anterior es almacenar la intancia única que asegura la función `SecurityManager::getInstance`, para de esta forma acceder a todos los recursos que la misma implementa. Luego se conforma el nombre del contenedor de seguridad y se le especifica al administrador a través de la función `SecurityManager::setSecurityContainer`, el cual se conforma en este caso por el valor definido en la propiedad `securityContainer` del fichero `./client_conf/auth.ini`, concatenado con el prefijo `SecurityContainer`. Una vez concluida la configuración preliminar se procede a ejecutar la función `SecurityContainer::checkUser` definida en este caso por la clase `SopAcaxiaSecurityContainer`.

Otro de los elementos que resultan utiles en el trabajo con el manejador de seguridad resultan las constantes que el mismo define, las cuales se describen a continuación:

- **ALL_ROLE**: hace referencia al rol que influye de forma general sobre todos los user y toma como valor por defecto `all`
- **ANONYMOUS_ROLE**: describe el identificador para el rol que puede tomar un usuario determinado en caso de no ser especificado manualmente, el cual toma como valor por defecto `anonymous`.
- **LOGGED_IN_ROLE**: toma como valor por defecto `loggedIn`
- **SESSION_AUTH_NAME**: define la estructura que se almacena en la variable de seccion una ves que el usuario se alla registrado, entre las propiedades mas

significativas se encuentra `__PHP_Incomplete_Class_Name`, la cual contiene el nombre de las clases del contenedor de seguridad utilizado.

En caso que se requiera implementar una función que permita conocer si un usuario se encuentra logiado en el sistema, se requiere el empleo de estas constates, para mayor comprensión observe el ejemplo que se muestra a continuación.

```
public function isLogin()
{
    $className = ucfirst( $config->securityContainer ) . 'SecurityContainer';

    if( isset( $_SESSION[ SecurityManager::SESSION_AUTH_NAME ] ) )
    {
        $auxContainer = get_object_vars(
            $_SESSION[SecurityManager::SESSION_AUTH_NAME]
        );

        if( isset($auxContainer['__PHP_Incomplete_Class_Name']) )
        {
            if( $auxContainer['__PHP_Incomplete_Class_Name'] != $className )
            {
                unset($_SESSION[SecurityManager::SESSION_AUTH_NAME]);
                return false;
            }
            return true;
        }else{
            if( $_SESSION [SecurityManager::SESSION_AUTH_NAME]
                instanceof $className )
                return true;
        }
    }
    return false;
}
```

En este ejemplo al igual que en el anterior, se conforma el nombre del contenedor de seguridad almacenado en el fichero de configuración de plugin. Luego se comprueba que se halla creado la sesión de usuario definida por la constante `SecurityManager::SESSION_AUTH_NAME`, en caso afirmativo se comprueba a través de la propiedad `__PHP_Incomplete_Class_Name` que el contenedor de seguridad corresponda con el que se había definido y en caso de no contener dicha propiedad se procedería a realizar dicha comprobación directamente a través del recurso `instanceof`.

Existen otros recursos provistos por esta clase tales como `SecurityManager::getUser` y `SecurityManager::getRoles`, permitiendo obtener información referente a que usuario está registrado en un instante de tiempo determinado, así como los roles asociados al mismo. Por otra parte los métodos `SecurityManager::setSecurityContainer`, `SecurityManager::getSecurityContainer` y `SecurityManager::clearSecurityContainer`, permiten obtener la instancia única de la clase, redefinir el contenedor de seguridad, así como asignarle valor `null` a dicho contenedor.

5.9.6 Priolus

Este *plugin* permite mapear sobre un área geográfica determinada un conjunto de estructuras anteriormente definidas, generalmente y propio de las peculiaridades del desarrollo en la UCID, estas son extraídas de sistema Estructura y Composición, mediante el acceso directo a la base de datos. Es importante destacar que la tabla que contenga dicha información debe incorporar un conjunto de características que le permitirían al servidor de mapa visualizar dicho contenido.

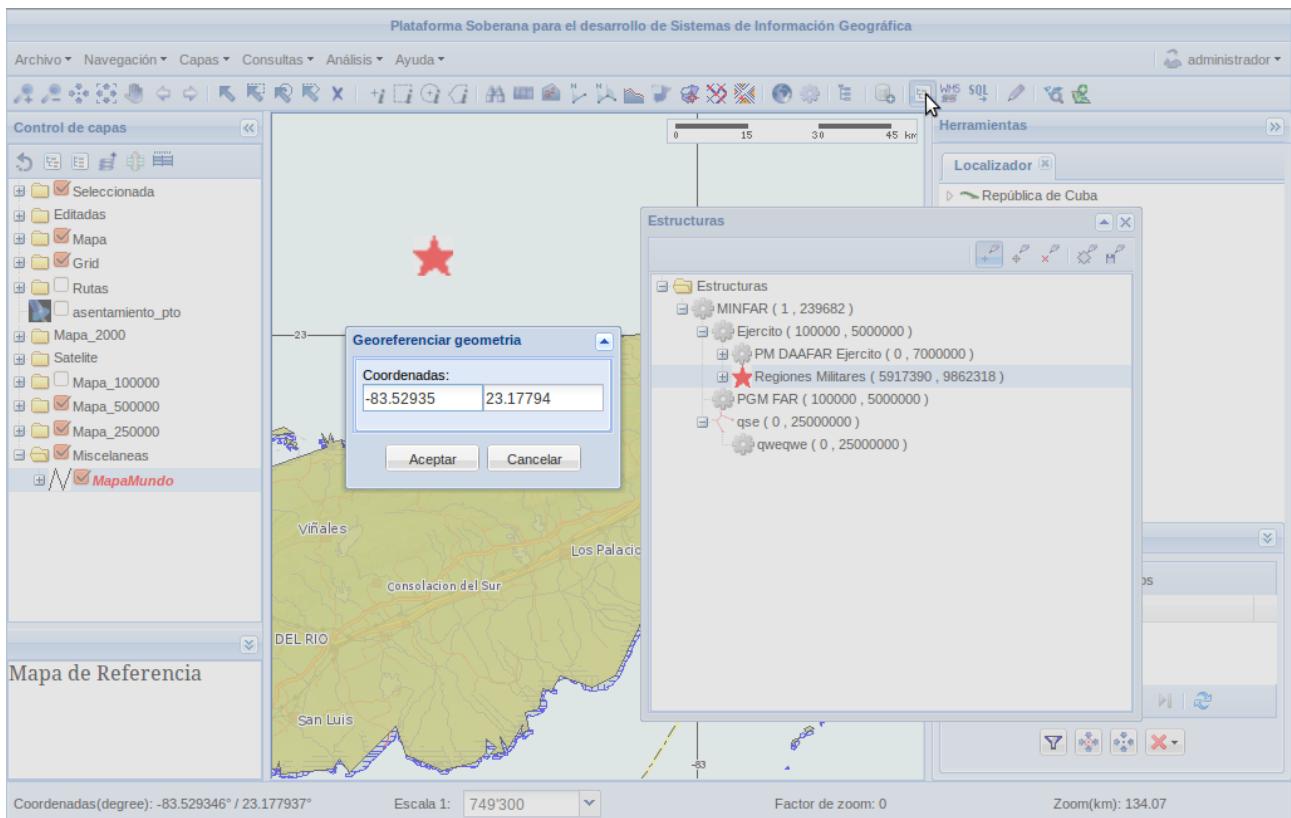


Ilustración 37 Modulo Priolus, aunque aparece con la denominación estructuras

Es muy común en las integraciones con otros subsistemas o sistemas externos, que estos contengan sus propias bases de datos y por tal motivo solo requieren que GeneSIG consuma la información que estos posee. Por tanto en aras de garantizar dicho objetivo debe ejecutarse el script que se muestra a continuación:

```
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN wkb_geometry geometry;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN imagen character(255) DEFAULT 'point'::character varying;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN oid oid DEFAULT 0;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN minscale integer DEFAULT 0;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN maxscale integer DEFAULT 0;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN latitud double precision DEFAULT 0;
ALTER TABLE mod_estructuracomp.nom_filaestruc ADD COLUMN longitud double precision DEFAULT 0;
```

Obsérvese que en el ejemplo anterior se asume como esquema a *mod_estructuracomp* y como tabla contenedora de la información necesaria a *nom_filaestruc*, sin embargo estos nombres pueden variar perfectamente.

Una vez compatibilizada la base de datos para que soporte la información necesaria para representar las estructuras, se requiere configurar algunas de las propiedades en la capa rectora de dicho contenido. Esta generalmente se denomina *estructura* y se encuentra ubicada a partir del directorio raíz de la plataforma en: */projects/<project-name>/server_conf/<map-name>/userMaps/<user-id>.map*, evidentemente aquellos directorios cuyo nombre se encuentra delimitado por los caracteres <> serian aquellos que varían en función de cual proyecto se encuentra en ejecución, así como el mapa y el usuario activo.

```
CONNECTION "host=10.12.170.159 user=postgres password=postgres dbname=SIDECK2"
CONNECTIONTYPE POSTGIS
```

Entre los parámetros que deben ser modificados se encuentra el *connection*, el cual describe atributos tales como:

- **host**: dirección *ip* del servidor de bases de datos.
- **dbname**: nombre de la base de dato.
- **user**: nombre del usuario mediante el cual se establecerá la conexión.
- **password**: contraseña definida para el usuario especificado.

Entre otro de los atributos que deben ser modificados se encuentra el *data*, el cual define la consulta que constituirá la tabla virtual que contendrá la información que será representada. A continuación se muestra un ejemplo aunque realmente esto varía de la estructura definida en la base de dato.

```
DATA "wkb_geometry FROM (SELECT nom_filaestruc.oid AS oid, nom_filaestruc.idfila AS  
idfila, nom_filaestruc.wkb_geometry AS wkb_geometry,  
CASE  
    WHEN nom_cargomilitar.dencargomilitar <> " THEN nom_cargomilitar.dencargomilitar  
    WHEN nom_cargocivil.dencargociv <> " THEN nom_cargocivil.dencargociv  
    WHEN nom_tecnica.dentecnica <> " THEN nom_tecnica.dentecnica  
    WHEN dat_estructura.denominacion <> " THEN dat_estructura.denominacion  
    WHEN dat_estructuraop.denominacion <> " THEN dat_estructuraop.denominacion  
END AS label, nom_filaestruc.imagen AS imagen, asText(nom_filaestruc.wkb_geometry) AS coords  
FROM mod_estructuracomp.nom_filaestruc  
LEFT JOIN mod_estructuracomp.dat_estructura OR (dat_estructura.idestructura = nom_filaestruc.idfila)  
LEFT JOIN mod_estructuracomp.dat_estructuraop OR (dat_estructuraop.idestructuraop = nom_filaestruc.idfila)  
LEFT JOIN mod_estructuracomp.nom_organo OR ((nom_organo.idorgano = dat_estructura.idorgano and  
nom_organo.idhomeav = nom_filaestruc.idhomeav) OR (nom_organo.idorgano = dat_estructuraop.idorgano and  
nom_organo.idhomeav = nom_filaestruc.idhomeav))  
LEFT JOIN mod_estructuracomp.dat_tecnica OR (dat_tecnica.idestructura = nom_filaestruc.idfila)  
LEFT JOIN mod_estructuracomp.nom_tecnica OR (nom_tecnica.idtecnica = dat_tecnica.idtecnica)  
LEFT JOIN mod_estructuracomp.dat_cargo OR (dat_cargo.idcargo = nom_filaestruc.idfila)  
LEFT JOIN mod_estructuracomp.dat_cargomtar OR (dat_cargomtar.idcargo = dat_cargo.idcargo)  
LEFT JOIN mod_estructuracomp.nom_cargomilitar OR (dat_cargomtar.idcargomilitar = nom_cargomilitar.  
idcargomilitar)  
LEFT JOIN mod_estructuracomp.dat_cargocivil OR (dat_cargo.idcargo = dat_cargocivil.idcargo)  
LEFT JOIN mod_estructuracomp.nom_cargocivil OR (dat_cargocivil.idcargociv = nom_cargocivil.idcargociv)  
) AS sTable USING UNIQUE oid USING SRID= -1"
```

5.9.7 Manipulación de Mensajes

La detención, control y seguimiento de errores es uno de los elementos claves en el desarrollo de un software, sobre todo si se trata de notificar a los clientes en caso de cualquier percance o surcunstancia en que se produca los mimos. Por tanto el mecanismo de gestión de mensajes juega un papel preponderante en este sentido, y en función de recrear una capa de abstraccion que facilite el trabajo con estos, GeneSIG provee un conjunto de métodos que facilitan en alguna medida su gestión y visualización.

ServerContext::addMessage es precisamente la implementación que brinda GeneSIG en su núcleo para la gestión de mensajes de todo tipo, para ello requiere que se le especifique un conjunto de parámetros, los cuales se describen a continuación:

- **plugin**: Se define como una instancia de alguna clase que se especialice de *PluginBase*, la cual hace referencia al plugin que lanza el mensaje.
- **messageld**: Define el identificador del mensaje
- **message**: Cadena de texto que contiene el cuerpo del mensaje a visualizar
- **channel**: Define el canal para el mensaje lanzado, toman el valor *Message::CHANNEL_USER* por devecto. Los posibles valores que el mismo puede asumir se detallan a continacion:
 - **Message::CHANNEL_USER**: Mensajes a nivel de usuario comunes

- **Message::CHANNEL_DEVELOPER:** Mensajes a nivel de desarrolladores, estos están más relacionados con el núcleo de la plataforma, los cuales tienen una afectación de carácter horizontal

A continuación se muestra un fragmento de código del *coreplugin image*, donde se valida que el directorio donde se depositan las imágenes de carácter temporal no exeda la cantidad prevista, el cual que ilustra en alguna medida como se puede poner en práctica dicho mecanismos.

```
$imgPath = $this->serverContext->getMapObj()->web->imagepath;
$imgCount = count( scandir($imgPath) );
if ( $imgCount > self::MAX_IMAGES_WARNING ) {
    $msg = sprintf(
        'Warning: you have a high number of generated '.
        'images (%u [warning threshold %u]). You should '.
        'run the following instruction in a command line '.
        'prompt: php cw3setup.php --clean',
        $imgCount,
        self::MAX_IMAGES_WARNING
    );
    $this->serverContext->addMessage(
        $this,
        'tooManyImg',
        $msg,
        Message::CHANNEL_DEVELOPER
    );
}
```

En caso que se requiera bajo determinadas circunstancias obtener el listado de mensajes generados en un instante de tiempo se debe utilizar el recurso *ServerContext::getMessages*.

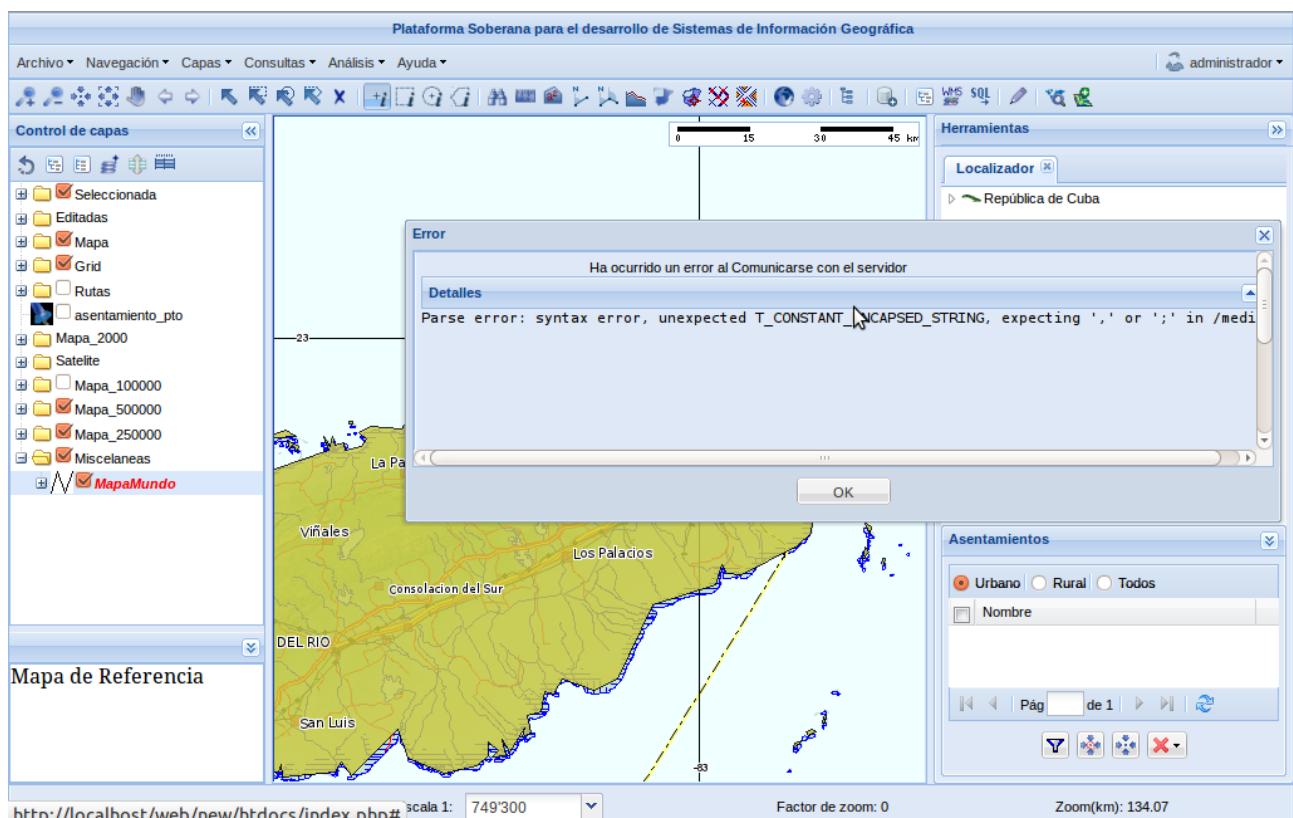


Ilustración 38 Interfaz de visualiza de mensajes sobre la plataforma GeneSIG

En la imagen anterior se puede observar la interfaz provista por la parte cliente de GeneSIG encargada de la visualización de dichos mensajes.

5.10 Tutorial

En este acápite se estará poniendo en práctica todos los elementos vistos anteriormente, tomando como base la confección de un proyecto denominado *ejem*, sobre el cual se estará desarrollando un modulo denominado *search*, además de adicionarle algunos de los que provee la propia plataforma.

5.10.1 Descargando el código fuente

Lo primero que debe hacerse antes de comenzar a desempaquetar el código fuentes es crear el espacio de trabajo y acondicionarlo en función de los permisos. Para ello podemos abrir una interfaz de líneas de comandos, donde se procederá a crear el directorio a través del recurso *mkdir*. Luego se direcciona la consola hasta el mismo y se especifica a partir de la raíz los privilegios de lectura, escritura y ejecución, esto se realiza a través de la aplicación *chmod*, para mayor comprensión véase la figura que se muestra a continuación.

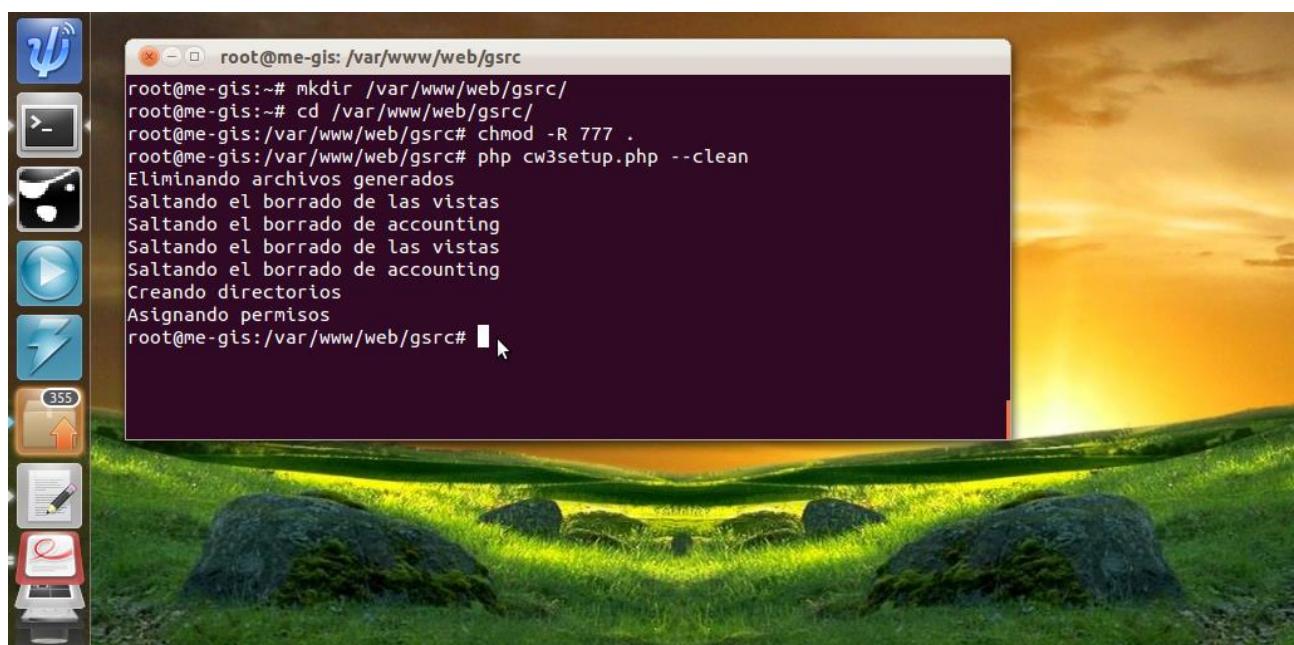


Ilustración 39 Comando de consola para crear el espacio de trabajo.

Nótese que la línea en que se utiliza el fichero *cw3setup.php*, es utilizada una vez descargado en el espacio de trabajo todo el código fuente del marco de trabajo. Esta permite limpiar la cache generada por GeneSIG, así que cada vez que se requiera actualizar algún cambio debe invocarse.

En función de iniciar el desarrollo sobre la plataforma GeneSIG, existen dos vías alternativas la primera y mas simple consiste en tener un archivo comprimido que contenga el fuente del marco de trabajo, al cual podemos acceder a través de la interfaz de líneas de comando de la siguiente forma:

```
tar xvzf gnesig.tar.gz
```

En caso que se requiera salvar todos los cambios realizados sobre el código fuente contenido en especio de trabajo definido anteriormente, se procedería a comprimir de la siguiente forma:

```
tar czvf gnesig.tar.gz --exclude=".svn" --exclude="*.*~" --exclude="*.tar.gz" .
```

Nótese que la opción *exclude* permite excluir aquellos directorios que coincidan con la expresión especificada, en este caso se obvian los fichero de extensión *.svn*, temporales y archivos comprimidos de extensión *.tar.gz*

Evidentemente el mecanismo anterior resuelve en gran medida el echo de comensar a desarrollar, así como el salvado de las nuevas modificaciones, incluso sería posible almacenar en un directorio un histórico de versiones comprimidas en caso que se requiera regresar a alguna versión específica. Sin embargo esto podría acarrear problemas de índole humana en la gestión de estos procesos y en función de esto se han desarrollado sistemas los automatizan. Por tanto lo más aconsejable es que trabajes contra un sistema de control de versiones.

El primer paso sería atraves de un cliente *svn*, en este caso se utilizar RapidSVN, crear una comprobación de nueva copia de trabajo, para mayor comprensión véase la figura que se muestra a continuación.

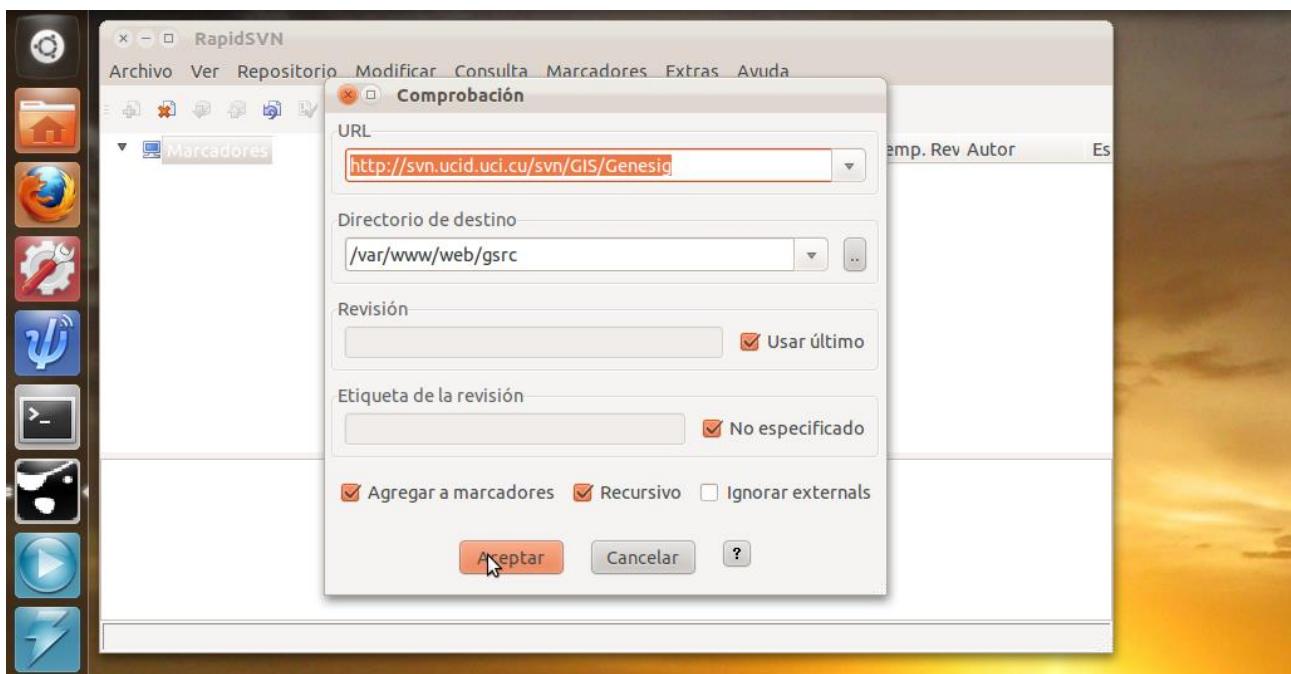


Ilustración 40 Primer paso en la descarga basada en un SVN

En la interfaz que muestra el cliente *svn* se le deben especificar evidentemente la dirección *url* donde se encuentra publicado el producto que en el caso de la línea seria: <http://svn.ucid.uci.cu/svn/GIS/Genesig>, así como el directorio del espacio de trabajo previamente creado.

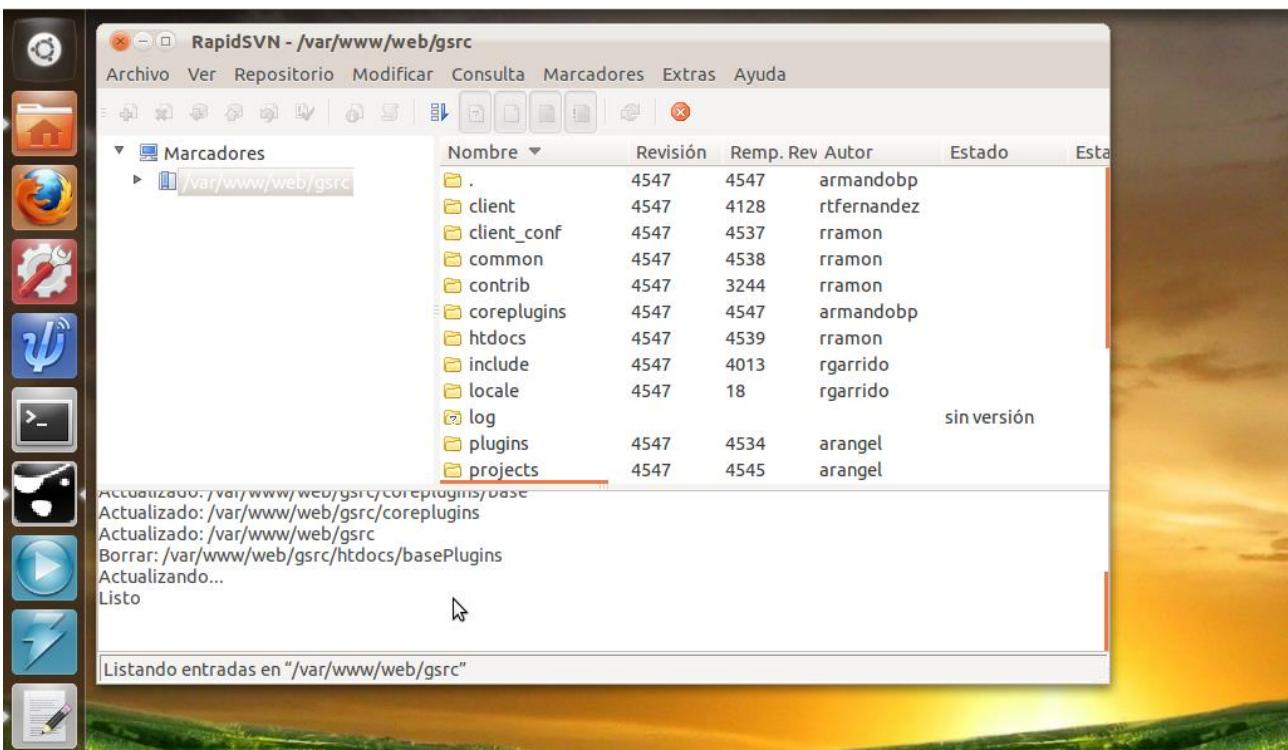


Ilustración 41 Segundo paso en la descarga basada en un SVN

Una característica peculiar del repositorio definido para trabajar con GeneSIG en el centro UCID, es que el fuente del *framework* esta separado del de los proyectos. Es por ello que se debe crear un directorio donde se procederá a descargar dicho fuente. Generalmente se crea dentro de un espacio de trabajo externo copiado luego el enlace simbólico, teniendo en cuenta que al utilizar alguna distribución de *GNU/LINUX* sería muy provecho el empleo de este recurso, de lo contrario se tendría que copiar físicamente en el directorio de proyecto especificado por la propia plataforma.

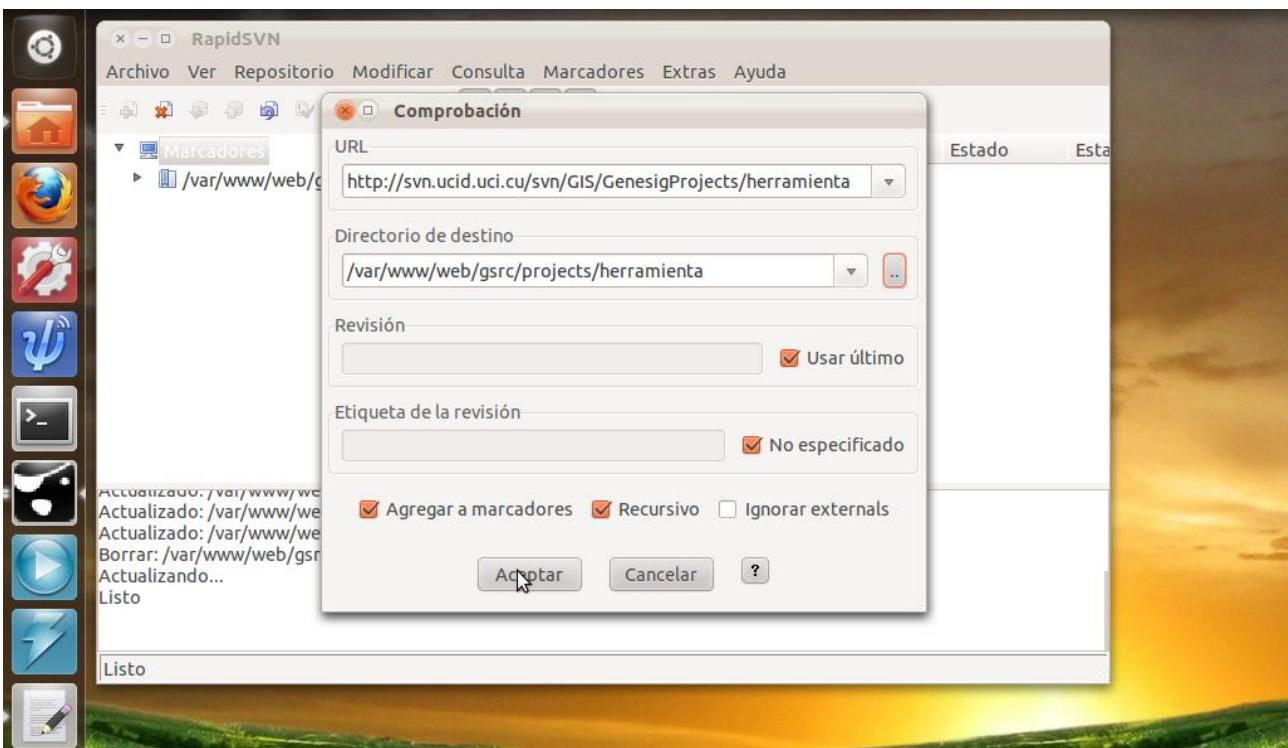


Ilustración 42 Tercer paso en la descarga basada en un SVN

Al igual que en el primer paso se le especifica el espacio de trabajo definido para el código fuente del proyecto a desarrollar, así como la *url* de publicación del mismo, que en el repo definido en el centro UCID para estos es: <http://svn.ucid.uci.cu/svn/GIS/GenesigProjects/>, en el cual se encuentra creado un proyecto que puede ser tomado como base para los demás y sirve de demostración, este se denomina *herramienta*.

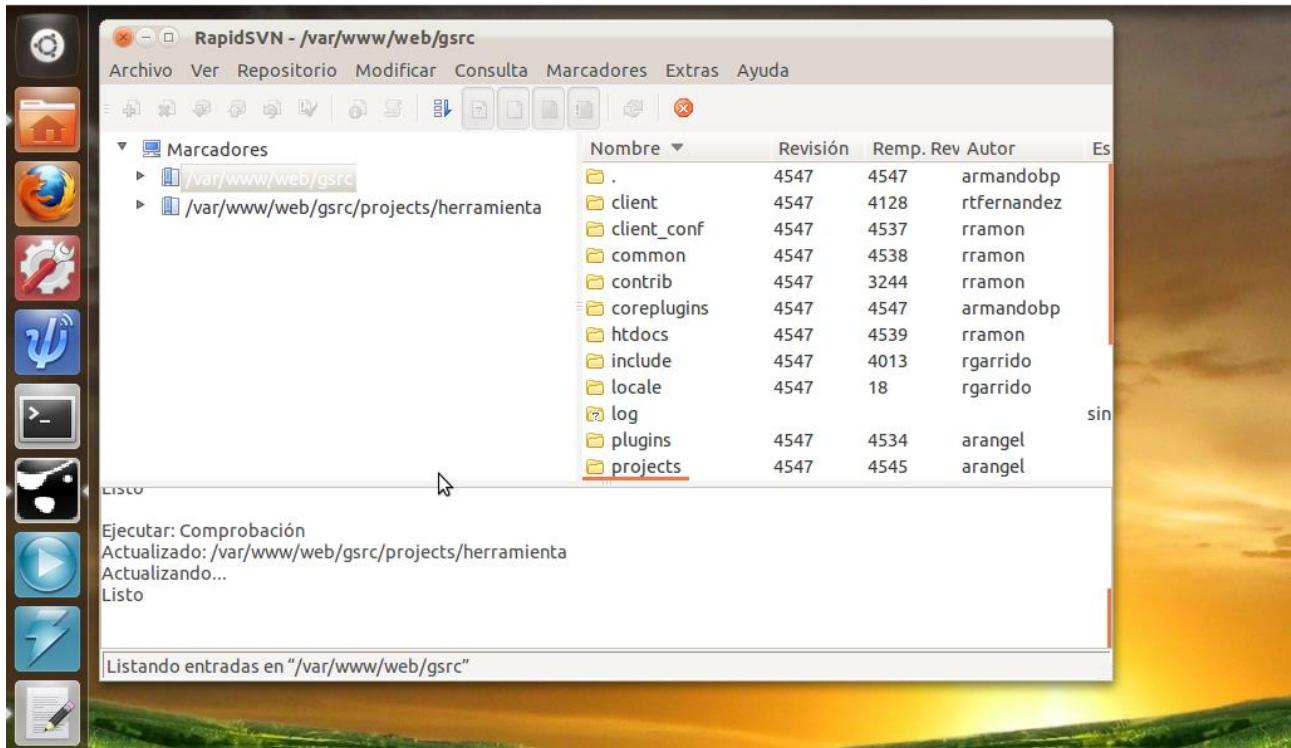


Ilustración 43 Cuarto paso en la descarga basada en un SVN

Una vez concluido estos procesos es recomendable que se le restablezca los permisos de lectura y escritura a los directorios definidos para los diferentes espacios de trabajo, luego se debe ejecutar el *cw3setup.php* en función de eliminar la cache en caso que se haya subido algo al servidor, evitando de esta forma respuestas no esperadas.

5.10.2 Creando un Proyecto

El proceso de confección de un proyecto se puede hacer de forma manual evidentemente, sin embargo esto podría acarrear también algún problema impulsado por el propio factor humano, sobre todo por lo tedioso que podría llegar a ser dicha tarea, teniendo en cuenta el cumulo considerable de ficheros de configuración, así como otros elementos tales como restricciones de nomenclaturas, etc. En consecuencia a esto fue desarrollado el módulos de configuración el cual permite de forma grafica e intuitiva automatizar estos procesos, potenciando un desarrollo ágil de software.

El primer paso en función de lograr nuestro objetivo, el cual está enfocado en la confección de un proyecto o personalización dirigida a un negocio determinado, seria acceder al sistema *herramienta* y accionar la opción de configuración, tal y como se muestra en la imagen que aparece a continuación.

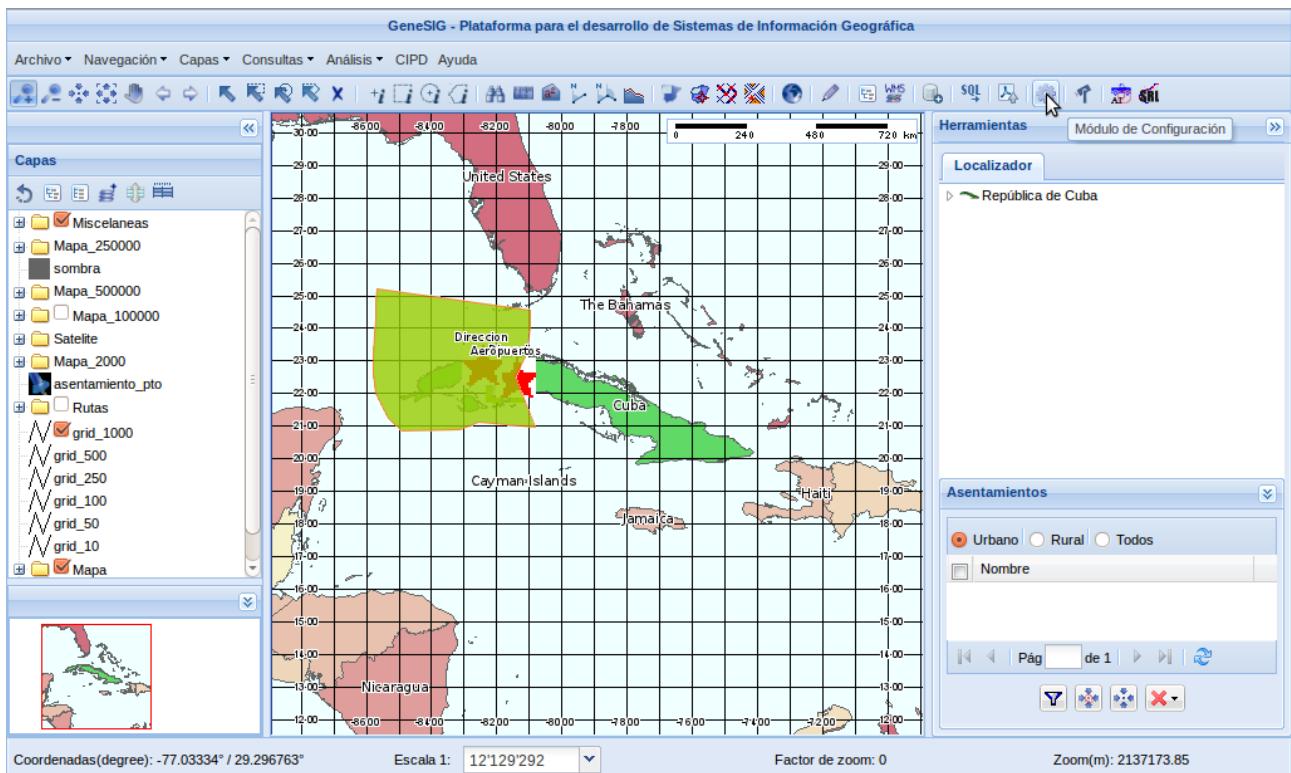


Ilustración 44 Paso 1 en la creación de un proyecto

El cual muestra a través de una ventana una serie de opciones que nos permitirán gestionar los elementos relacionados con proyectos, módulos o agrupaciones de plugins, usuarios, roles, incluso puede ser especificados que acciones por Plugin se desea visualizar en la barra de herramientas.

Para crear físicamente el proyecto es necesario acceder a la opción *Proyecto* ubicada en la región izquierda de la interfaz, permitiendo observar el listado de proyectos activos en nuestro espacio de trabajo, así como la facilidad de gestionar los cambios sobre los mismos.

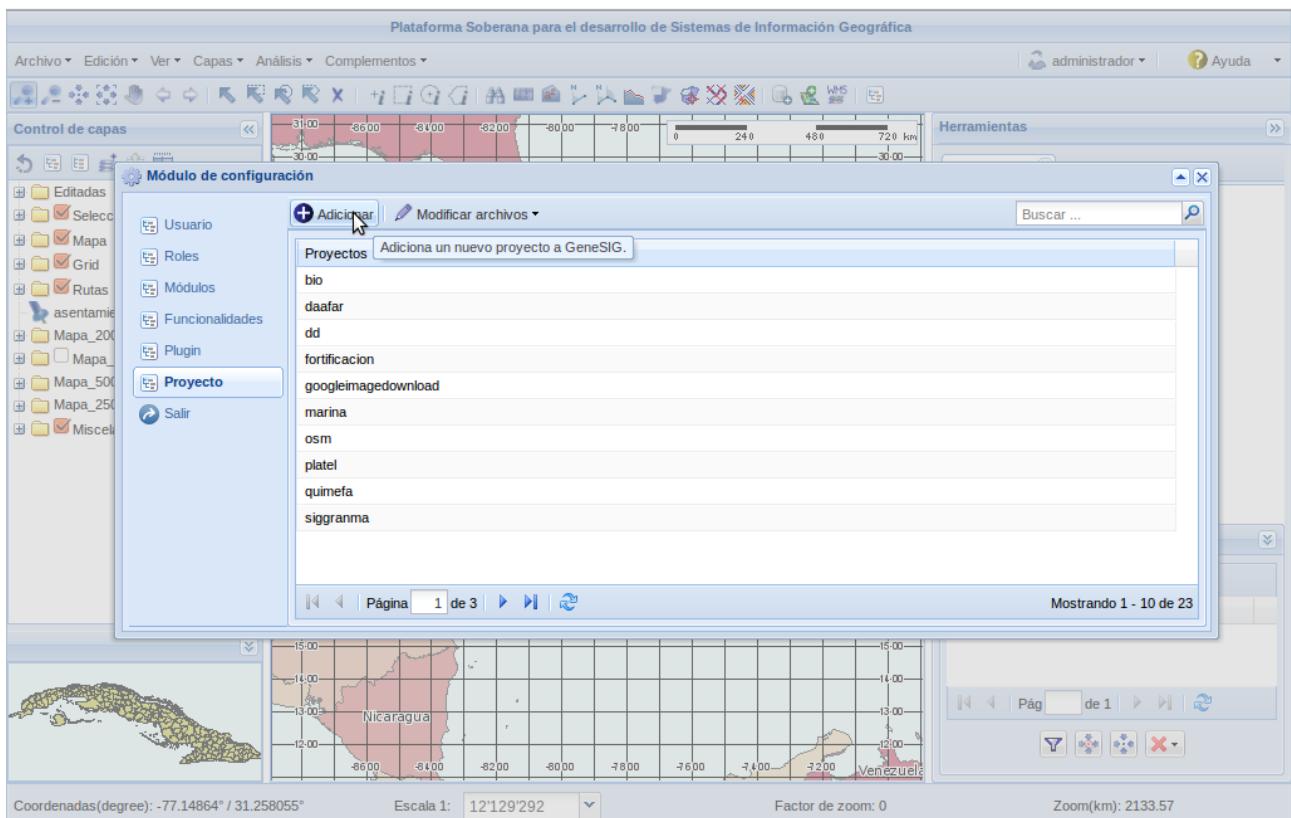


Ilustración 45 Paso 3 en la creación de un proyecto

Una vez presionado el botón adicionar posicionado en la parte superior de la ventana, se muestra una interfaz con un formulario en el cual debe ser especificado el identificador semántico para el nuevo proyecto, así como la denominación del proyecto que se tomara como padre, lo que implica que se asumiría las configuraciones y plugin que el mismo contenga, no obstante se puede escoger entre realizar una especialización de tipo total o personalizada a través del grubbox que se muestra en la interfaz que aparece a continuación.

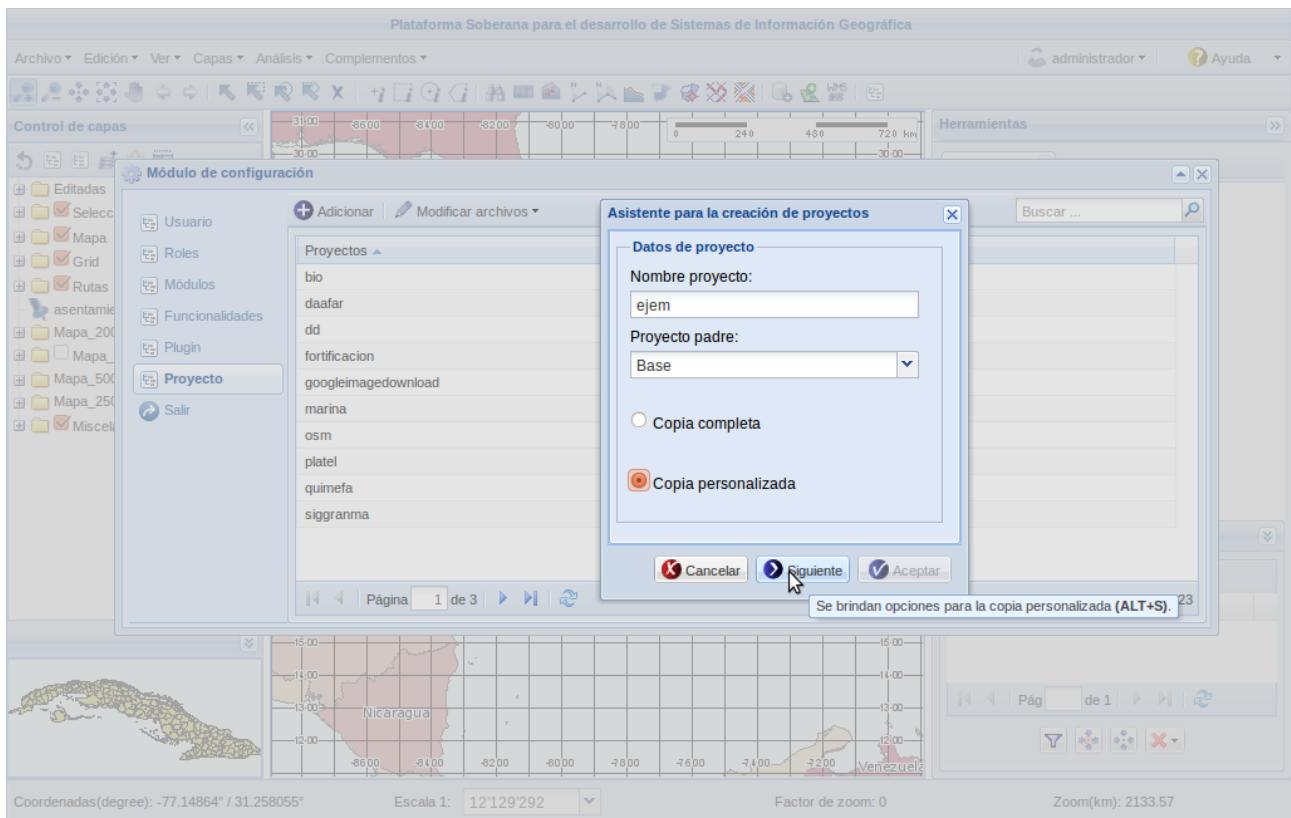


Ilustración 46 Paso 4 en la creación de un proyecto

Una vez presionado el botón *Siguiente* se muestra una interfaz que permite organizar los plugin que se encuentran asignados o no al nuevo proyecto, los cuales pueden ser estructurados en función de las necesidades a través de los botones destinados para este fin, así como la acción de clic derecho generándose un menú contextual en función de lograr dicho objetivo.

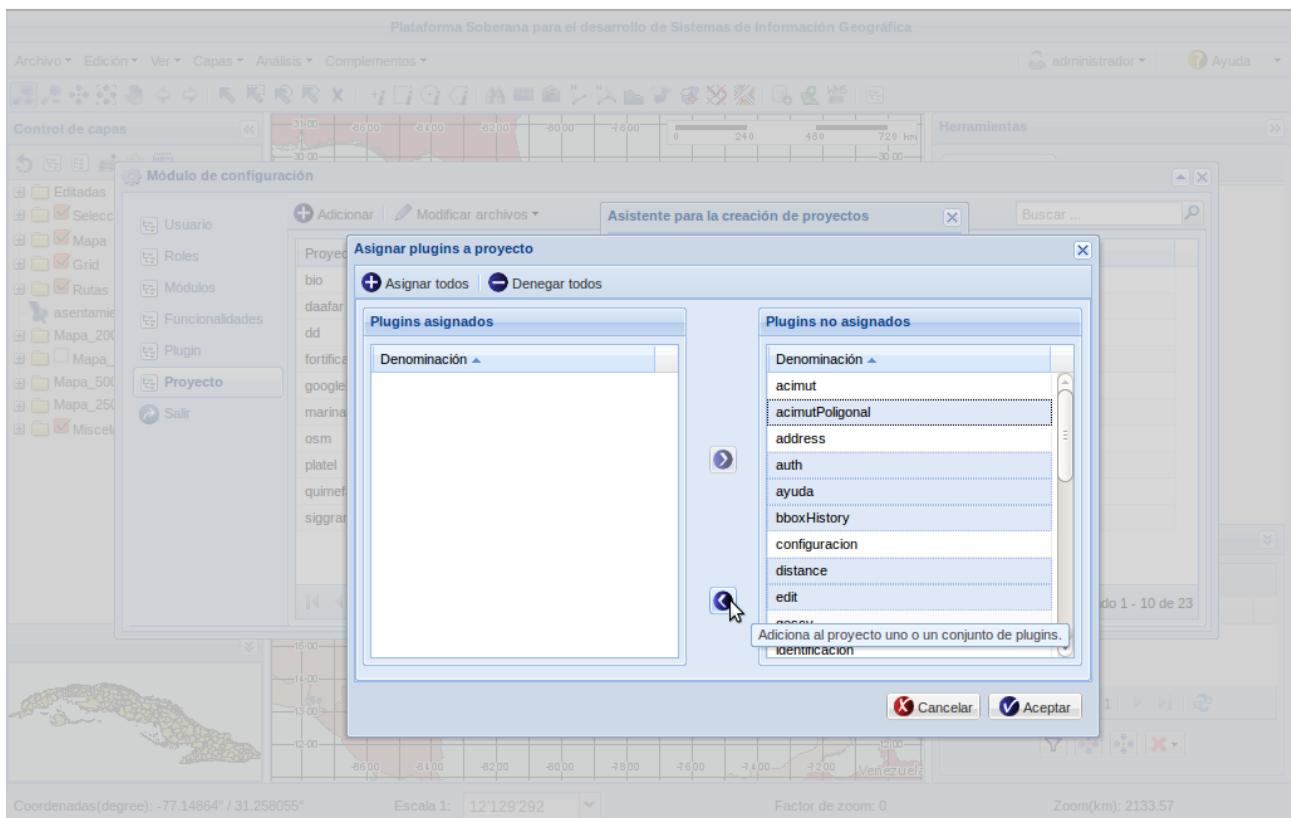


Ilustración 47 Paso 5 en la creación de un proyecto

Al concluir, se procede a acceder al proyecto denominado ejem a través de la url, especificándose en la propiedad *project*. También es posible anclar el mismo en función de que sea el que se invoque por defecto una vez iniciado el sistema, para mayor comprensión refiérase al acápite 5.6 *Controladores Frontales*.

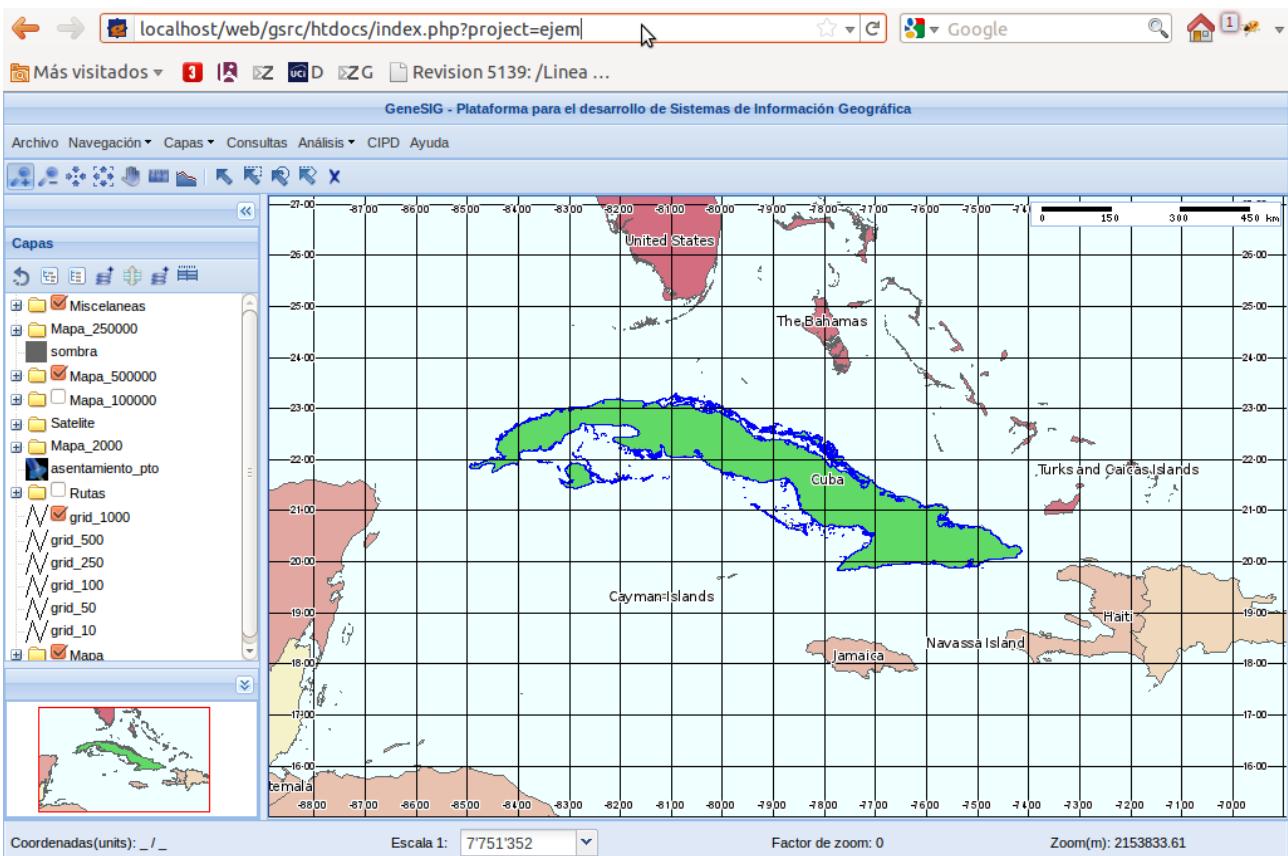


Ilustración 48 Paso 7 en la creación de un proyecto

5.10.3 Adicionando un Plugin

Este modulo también permite adicionar un plugin a un proyecto de forma gráfica e intuitiva sin que los desarrolladores requieran conocimientos avanzados en cuanto a los elementos que intervienen en la configuración de la plataforma, además agiliza dicho proceso. No obstante si se desea profundizar en el tema remítase al acápite 5.2 *Configuración*.

En función de lograr el objetivo planteado el sistema muestra una ventana con una amplia gama de opciones que al accionar el tabulador denominado *Plugin*, muestra todas la posible acciones a alizar, así como la información básica asociada a los plugin que se encuentran disponibles.

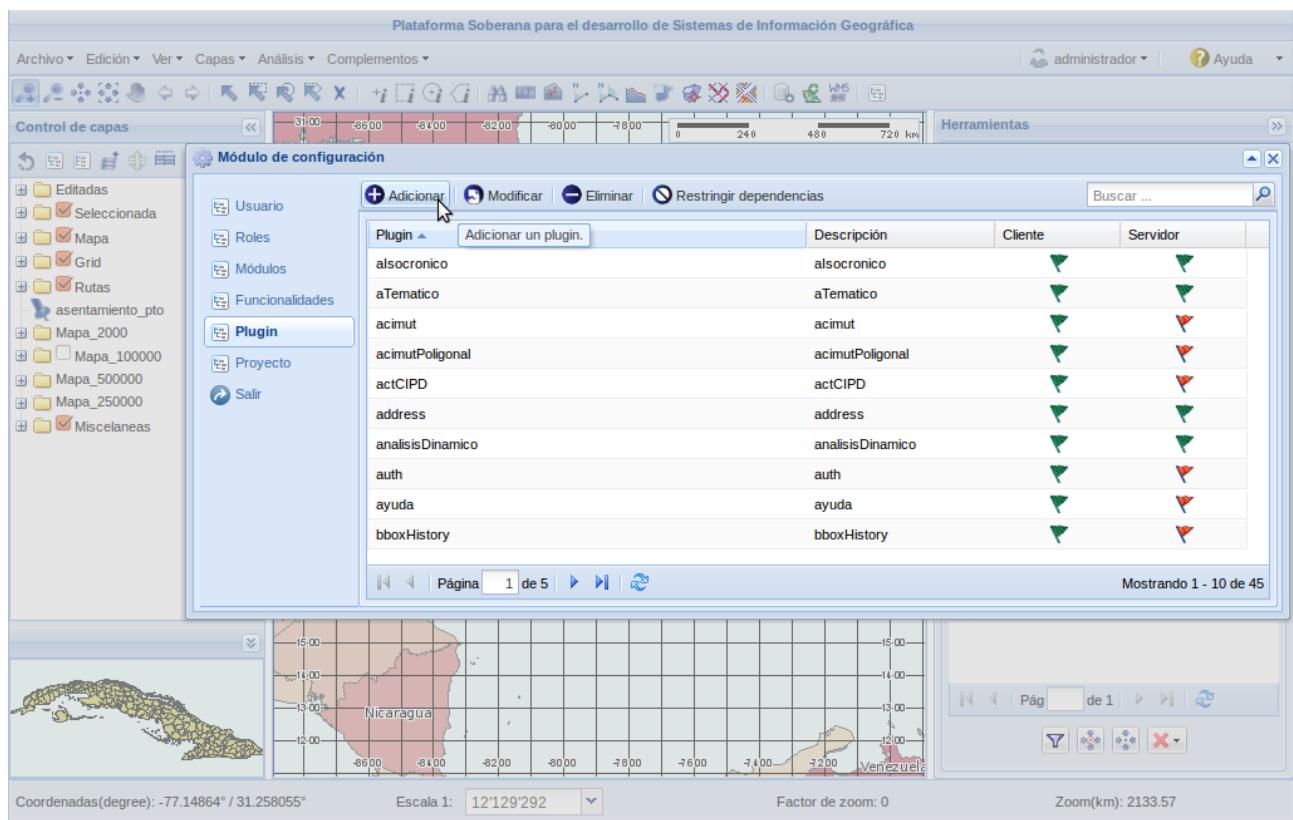


Ilustración 49 Paso 1 en función de adicionar un Plugin

Una vez presionado el botón denominado *Adicionar*, se muestra un formulario donde se debe especificar los elementos necesarios en función de la incorporación del nuevo plugin, entre los que se encuentra la denominación, la descripción y si contiene tanto partes servidores como clientes.

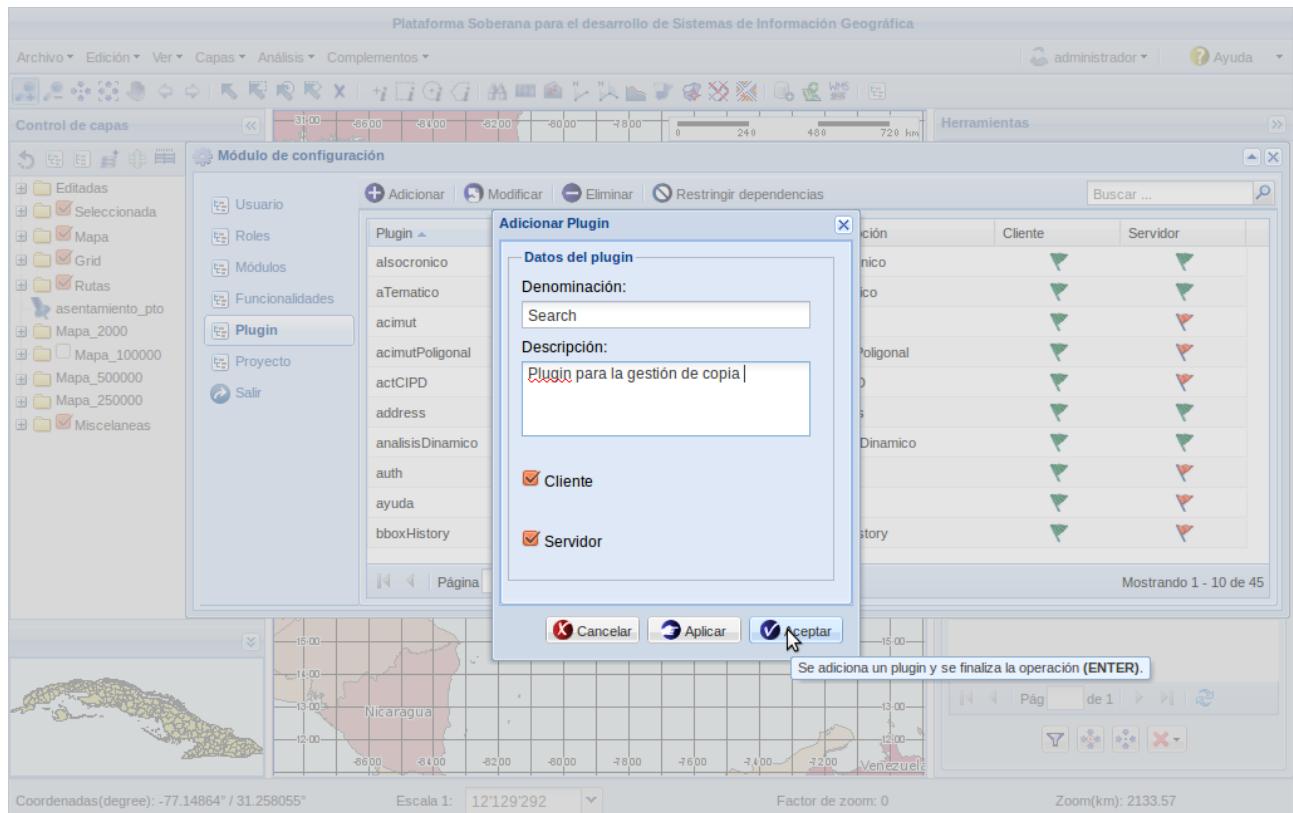


Ilustración 50 Paso 2 en función de adicionar un Plugin

Una vez invocada la operación que permite almacenar los dato especificados, el propio sistema notificaría el estado de la misma. En caso de ocurrir algún error se recomienda que revisen los permisos definidos para el directorio raíz de la plataforma.

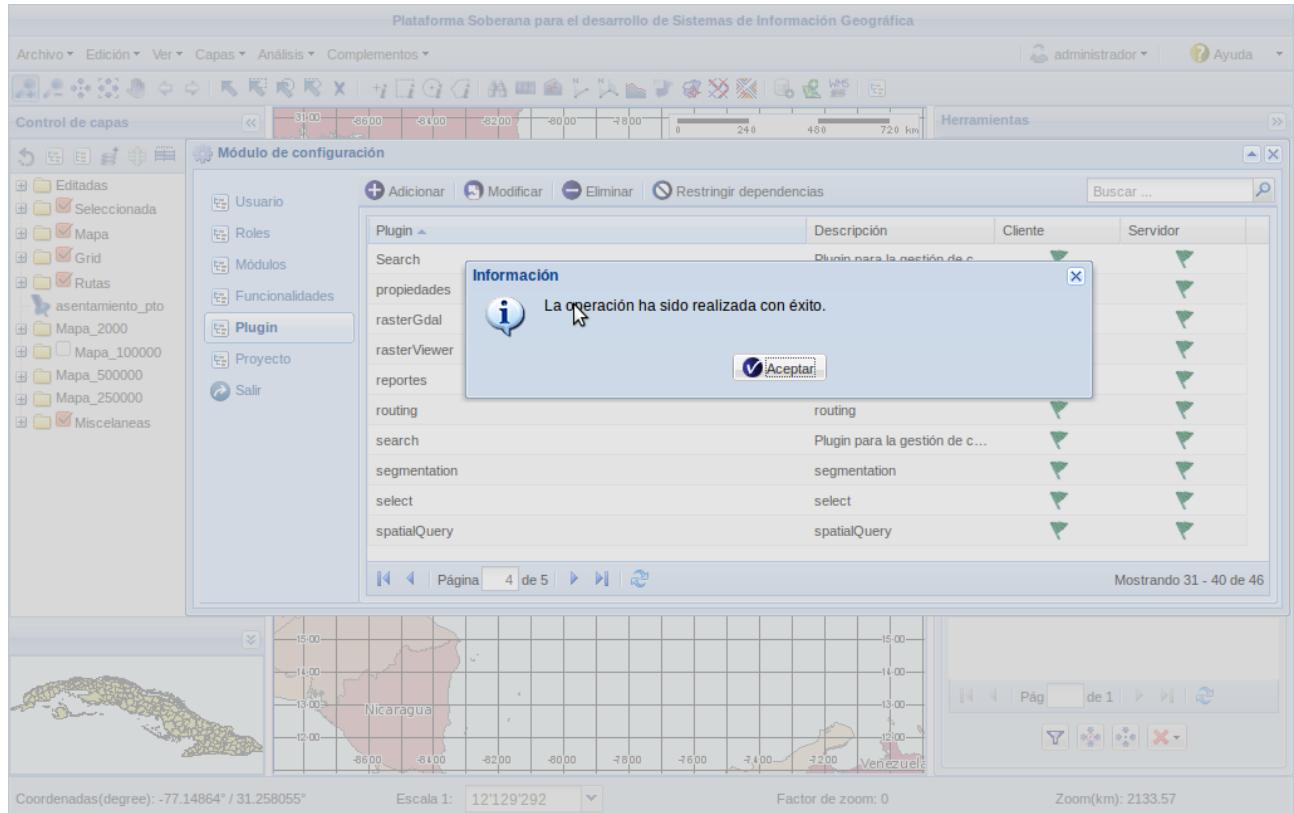


Ilustración 51 Paso 3 en función de adicionar un Plugin

Una vez registrado un plugin determinado es necesario tener en cuenta que estos se componen por un conjunto finito de *Funcionalidades*, las cuales son visualizadas en la barra de herramienta y en el menu contextual. Por otra parte como macro concepto de agrupación se maneja el término de *Modulo*, el cual permite encapsular un conjunto de plugins impactando en la organización de estos en los distintos componentes GUI que se emplean para su representación, para mayor comprensión véase la figura que se muestra a continuación.

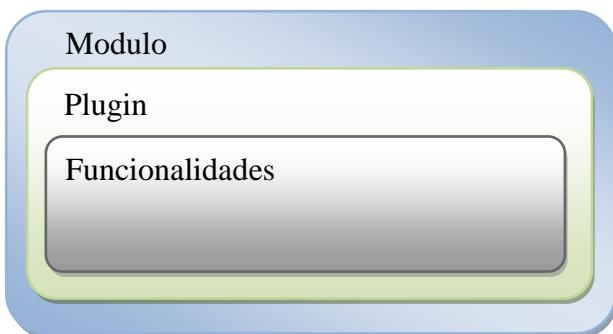


Ilustración 52 Jerarquía del esquema organizacional de las herramientas en la GUI

Teniendo en cuenta lo antes mencionado se proveen recursos que permiten de una forma intuitiva gestionar este macro concepto y en función de esto se crea una interfaz que muestra el listado actual de estas así como las operaciones que pueden ser realizadas sobre estos, aunque existen algunos predefinidos que pueden ser utilizados tales como: Navegación, Selección, Consulta, Análisis, Misceláneas, Capas, etc.

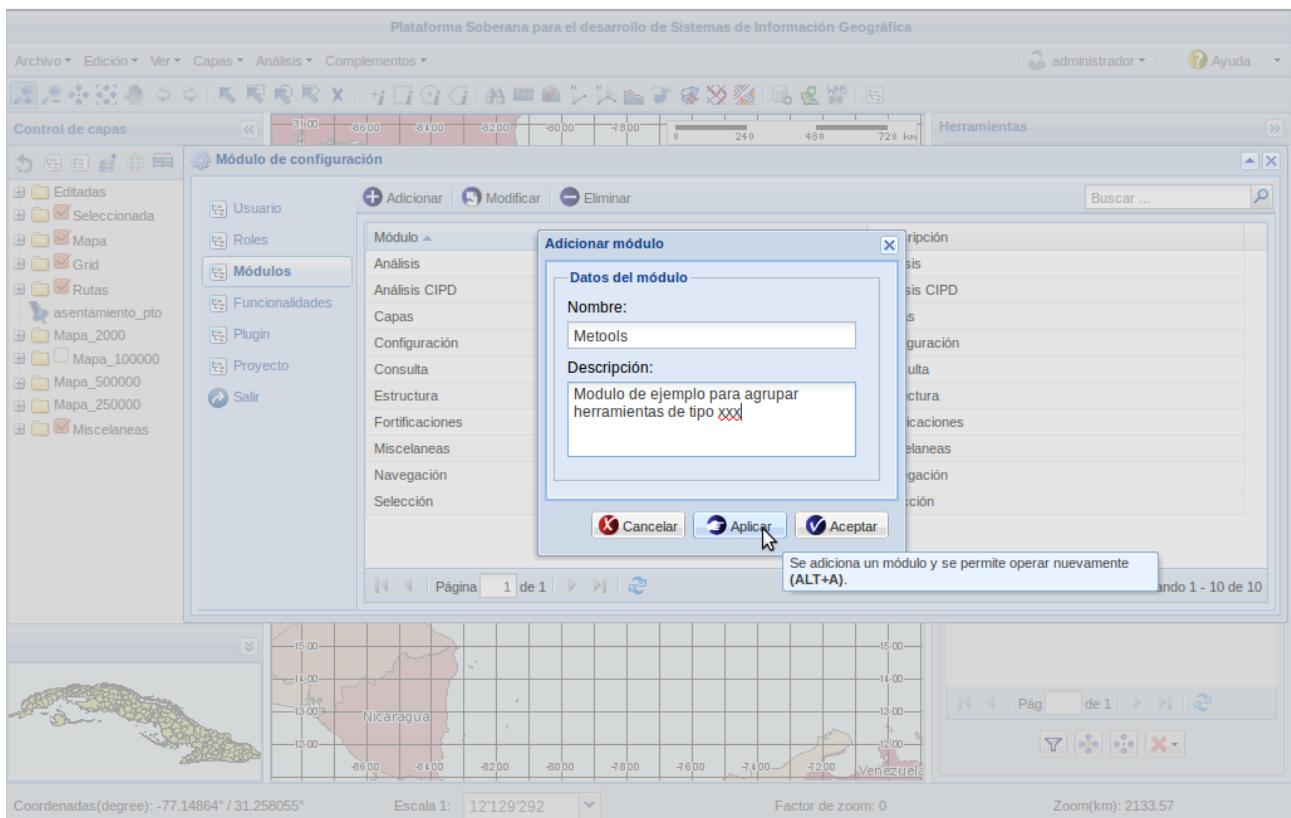


Ilustración 53 Paso 4 en función de adicionar un Plugin

Una vez defino el modulo en que será incluido nuestro plugin entonces se procede a describir cada una de las funcionalidades que lo componen. Para ello se accede a la vista que permite obtener la información básica asociada a estas así como las operaciones que pueden ser realizadas sobre las mismas.

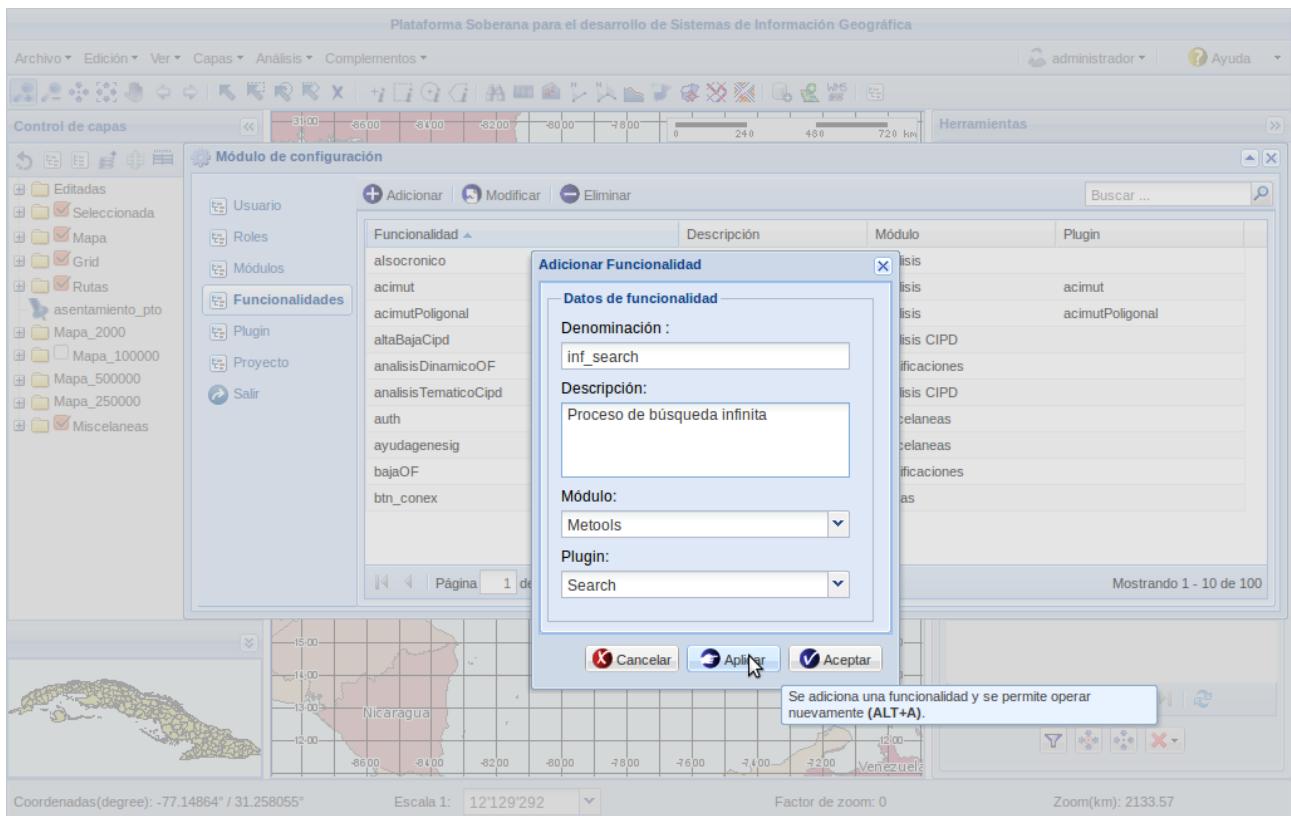


Ilustración 54 Paso 5 en función de adicionar un Plugin

Obsérvese que en el formulario definido para adicionar una nueva funcionalidad, es clave especificar la denominación del plugin a la que esta pertenece así como modulo al cual se subordina dicho plugin. Al finalizar este proceso se deben comenzar a gestionar los permiso asignados para los distintos roles que conforman el sistema, teniendo en cuenta que si no se habilitan para algún usuario determinado, ninguna de las funcionalidades serian mostradas en la interfaz grafica de usuario, lo que seria equivalente a la inexistencia del plugin en si mismo.

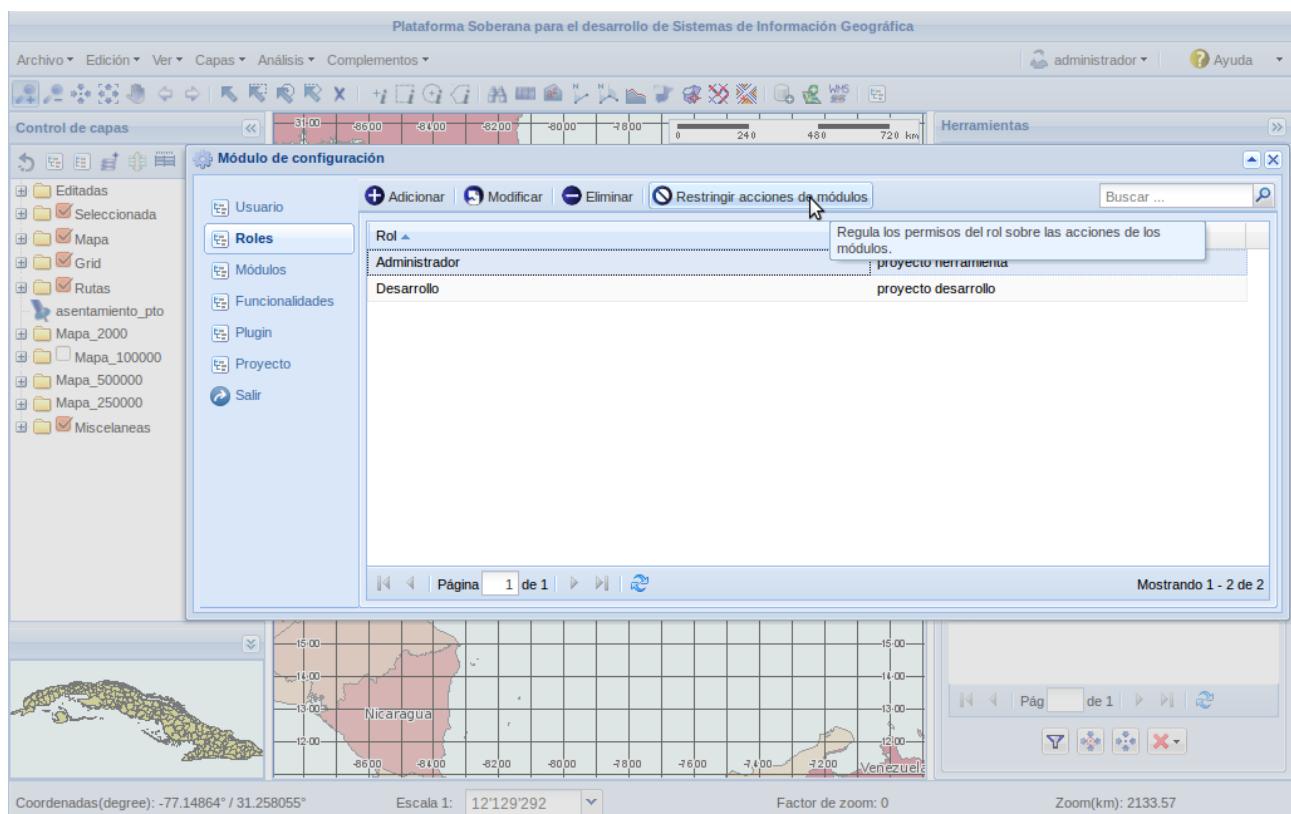


Ilustración 55 Paso 6 en función de adicionar un Plugin

A través de la operación definida como *Restringir acciones de modulos* es posible gestionar las funcionalidades que será habilitadas a o no para determinado rol, estas se muestran en una interfaz agrupadas por modulos, las cuales pueden ser autorizadas o denegadas a través de los botones definidos para tal objetivo o un menu contextual sobre la selección previamente seleccionada, tal y como se muestra en la imagen que aparece a continuación.

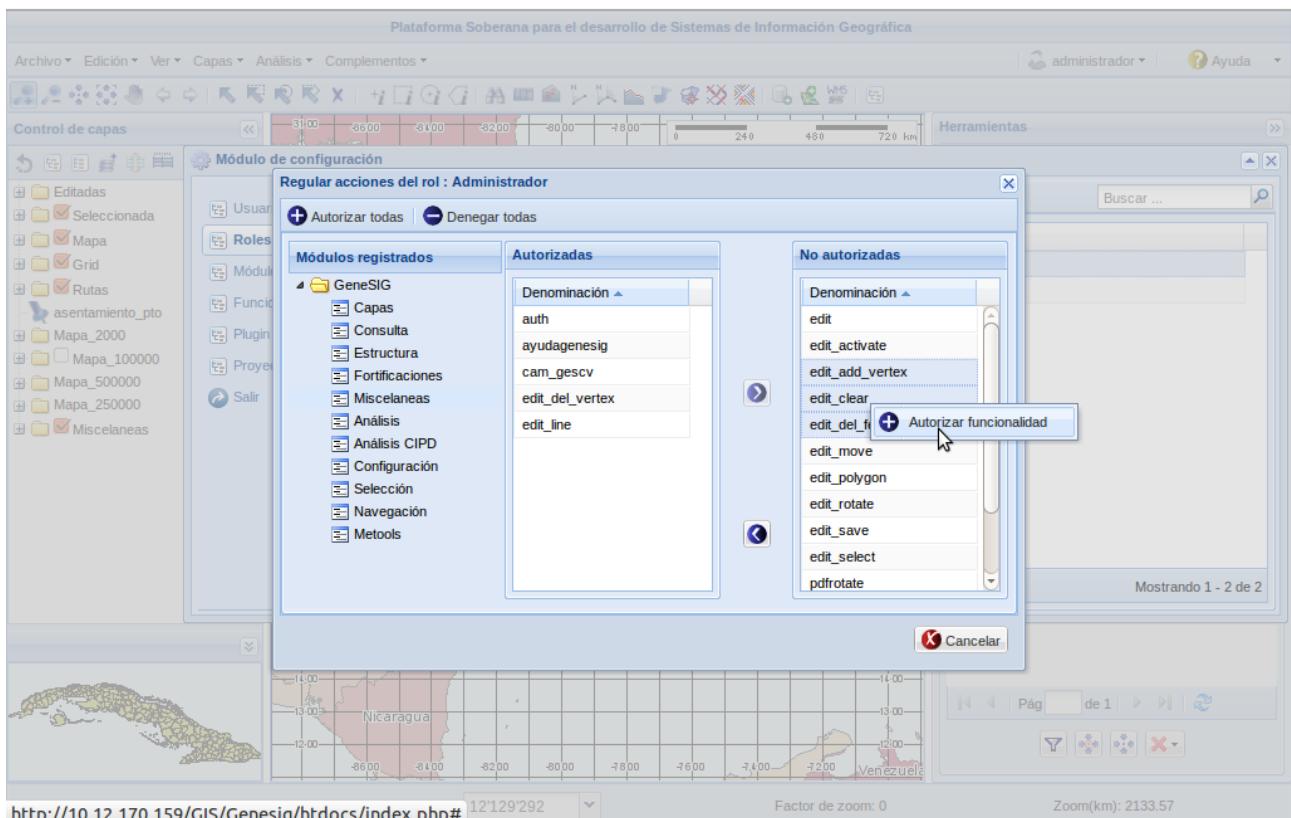


Ilustración 56 Paso 7 en función de adicionar un Plugin

Finalmente asegúrese de que los usuarios que requieran utilizar dicha funcionalidades pertenezcan al rol definido en el paso anterior y en función de solventar esta necesidad, este modulo provee una interfaz que permite gestionar los elementos relacionados con las cuentas de usuario, tal y como se muestra e la imagen que aparece a continuacion.

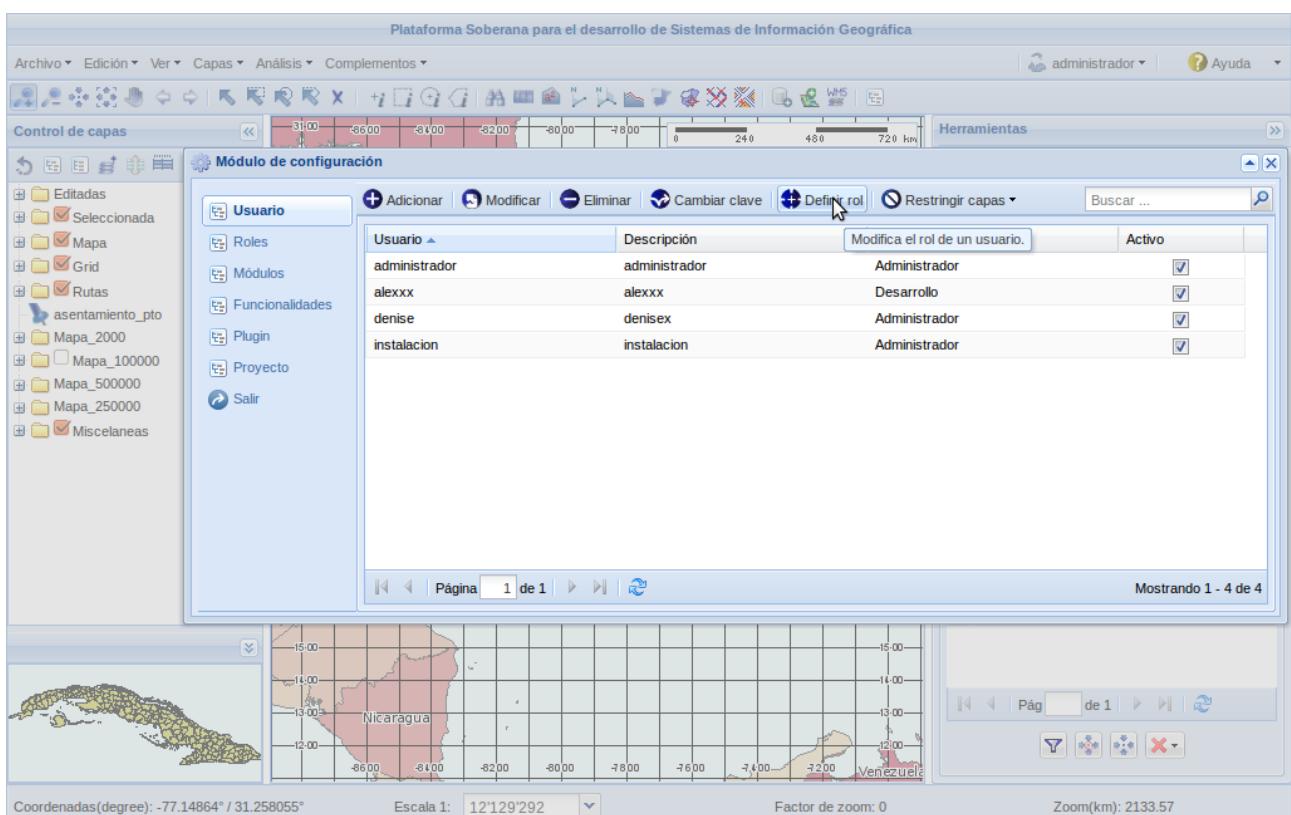


Ilustración 57 Paso 8 en función de adicionar un Plugin

5.10.4 Desarrollando un Plugin

En este acápite se procederá a describir el proceso de implementación de un plugin denominado *search*, en el cual se pronen en practico todos los elementos que se estuvieron abordando en acápitres anteriores. Por tanto se hace uso del API definido por GeneSIG para interactuar con el mapa, permitiendo identificar un listados puntos, los cuales serian enviados al servidor para que puedan ser procesados y de esta forma arrojar una respuesta al usuario.

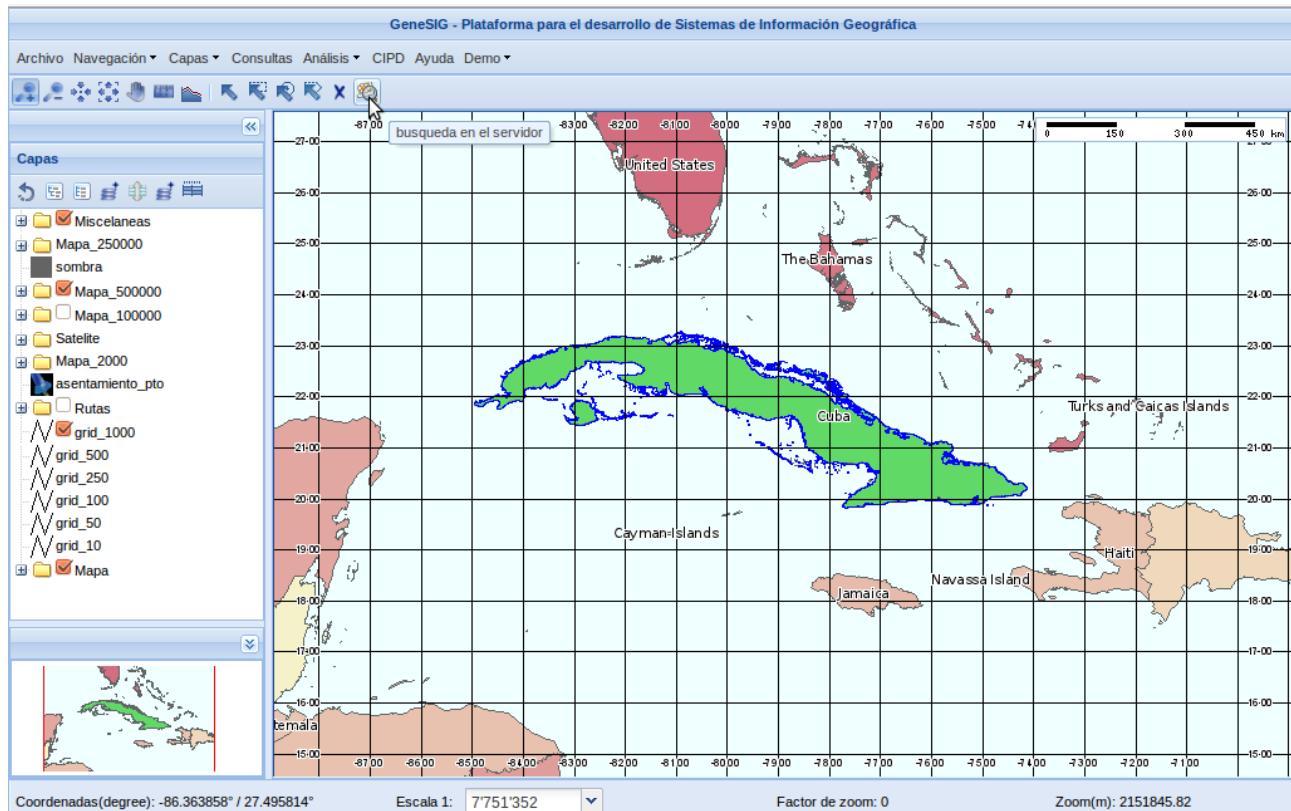


Ilustración 58 Accediendo al plugin Search a través del toolbar

Un elemento importante a destacar es que se asume la previa creación del proyecto el cual fue denominado *ejem* y que estaría ubicado en el directorio *projects* como bien se había descrito en acápitres anteriores, así como la creación de la estructura básica de este plugin que se toma como caso de estudio, tal y como se muestra en la imagen que aparece a continuación.

projects	3 elementos carpeta	mié 18 ene 2012 10:29:20 CST
ejem	3 elementos carpeta	mié 18 ene 2012 10:29:49 CST
client_conf	32 elementos carpeta	mié 18 ene 2012 10:29:22 CST
plugins	3 elementos carpeta	mié 18 ene 2012 10:29:49 CST
distance	4 elementos carpeta	mié 18 ene 2012 10:29:49 CST
perfil	4 elementos carpeta	mié 18 ene 2012 10:29:49 CST
select	5 elementos carpeta	mié 18 ene 2012 10:29:49 CST
server_conf	5 elementos carpeta	mié 18 ene 2012 10:29:22 CST
herramienta	3 elementos carpeta	mié 18 ene 2012 09:45:53 CST
siggrama	3 elementos carpeta	mié 18 ene 2012 09:40:18 CST

Ilustración 59 Taxonomía del proyecto ejem

En función de minimizar la complejidad de desarrollo se propone dividirlo en dos fases

una en la que se desarrolla la parte cliente, la cual tiene implicación directa en la interfaz gráfica de usuario y en una segunda iteración en la que se desarrollaría la parte servidor donde se procesaría la lógica de negocio así como la interacción con el servidor de mapas.

search	4 elementos carpeta
client	1 elemento carpeta
common	2 elementos carpeta
htdocs	3 elementos carpeta
css	1 elemento carpeta
gfx	1 elemento carpeta
js	5 elementos carpeta
ejschart	7 elementos carpeta
search.ajax.js	2,2 KiB programa en JavaScript
search.aux.js	7,2 KiB programa en JavaScript
search.gui.js	4,3 KiB programa en JavaScript
search.tool.js	3,3 KiB programa en JavaScript
server	1 elemento carpeta

Ilustración 60 Estructura de la parte cliente del modulo Search

Teniendo en cuenta las peculiaridades de la plataforma, se recomienda que dividir la implementación cliente en varios ficheros, en los cuales se pueda identificar fácilmente las distintas responsabilidades, las cuales generalmente se encuentran relacionadas con la gestión de las peticiones al servidor, la generación de interfaces gráficas de usuario, el comportamiento de la herramienta, entre otros. A continuación se procederá a describir cada uno de estos.

Contenido del fichero *search.ini*, el cual contiene la configuración relacionada con la parte cliente del modulo.

```
[views]
compresJS = 0
staticLoad[] = search.ajax.js

dynamicLoad[] = ejscart/dist/EJSChart.js
dynamicLoad[] = ejscart/JSChart.js
dynamicLoad[] = search.aux.js
dynamicLoad[] = search.gui.js
dynamicLoad[] = search.tool.js

cssLoad[] = ../js/ejscart/dist/EJSChart.css
cssLoad[] = style.css

[Menu]
demo.id = "demo"
demo.name = "demo"
demo.text = "Demo"

demo.calculo.id = "democalculoid"
demo.calculo.name = "democalculoname"
demo.calculo.text = "Cálculo"

demo.calculo.seach.id = "searchmenuid"
demo.calculo.seach.name = "searchmenuname"
demo.calculo.seach.text = "Busqueda Sobre el Terreno"
demo.calculo.seach.iconCls = "search-ico"
```

Código perteneciente al fichero *search.tool.js*, responsable del comportamiento del sistema una vez que haya sido activada la herramienta.

```
Map.prototype.perfil = function(aDisplay)
{
    var _this = this;
```

```

this.resetMapEventHandlers();
this.setCurrentLayer('perfil');
this.getDisplay(aDisplay).setTool('draw.line');
this.getDisplay(aDisplay).useSnapping = false;

this.calculateDistance = function(_pto1, _pto2)
{
    switch(AjaxPlugins.StaticTools.tDist){
        case 0: return GenericGeometryCalculation.EuclidianDistanceCalculation(_pto1, _pto2);
        case 1: return GenericGeometryCalculation.SphericalDistanceCalculation(_pto1, _pto2, true);
    }
}

this.onMove = function( x, y )
{
    if(_this.polilinea)
    {
        Actual_x = x;
        Actual_y = y;
    }
    if (_this.geoTag)
    {
        Actual_x = x;
        Actual_y = y;
    }
};

this.onClic = function(aFeature)
{
    _this.polilinea = aFeature;
};

this.onNewFeature = function(aFeature)
{
    this.onToolUnset();
    AjaxPlugins.Perfil.Aux.clearTravel("drawTravelPerfile");
};

this.onFeatureInput = function(aFeature)
{
    aFeature.operation = "";
    var lst = aFeature.vertices;
    if(lst.length > 0){
        AjaxPlugins.Perfil.Aux.Coord = new Array();
        AjaxPlugins.Perfil.Aux.Limpia_Series();
        lst[0].d = 0;
        AjaxPlugins.Perfil.Aux.Coord.push( [lst[0].x, lst[0].y, lst[0].d] );
        for(var i=1; i<lst.length; i++)
        {
            lst[i].d = lst[i-1].d + _this.calculateDistance(lst[i-1], lst[i]);
            AjaxPlugins.Perfil.Aux.Coord.push( [lst[i].x, lst[i].y, lst[i].d] );
        }
    }
    CartoWeb.triggerLight("Perfil.Perform");
};

this.onToolUnset = function()
{
    var usePanAccessKey = xGetElementById('usePanAccessKey');
    if (usePanAccessKey) {
        if (xGetElementById('usePanAccessKey').value != 'true') {
            this.getDisplay(aDisplay).clearLayer('perfil');
            this.onCancel();
        }
    }
    AjaxPlugins.Perfil.Aux.lineShow = this.getDisplay(aDisplay);
};

this.onCancel = function(aFeature)

```

```

    {
        this.distanceTag.style.display = "none";
    };
}

```

El objetivo principal de este script consiste que en cuanto se active la herramienta posibilite dibujar en forma trazado una serie de putos que hallan sido seleccionados por el usuario, a su vez estos deben ser almacenados de forma consecutiva con sus respectivas distancias acumuladas en función de ser enviados al servidor donde serán procesados, tal y como se muestra en la imaje que aparece a continuación.

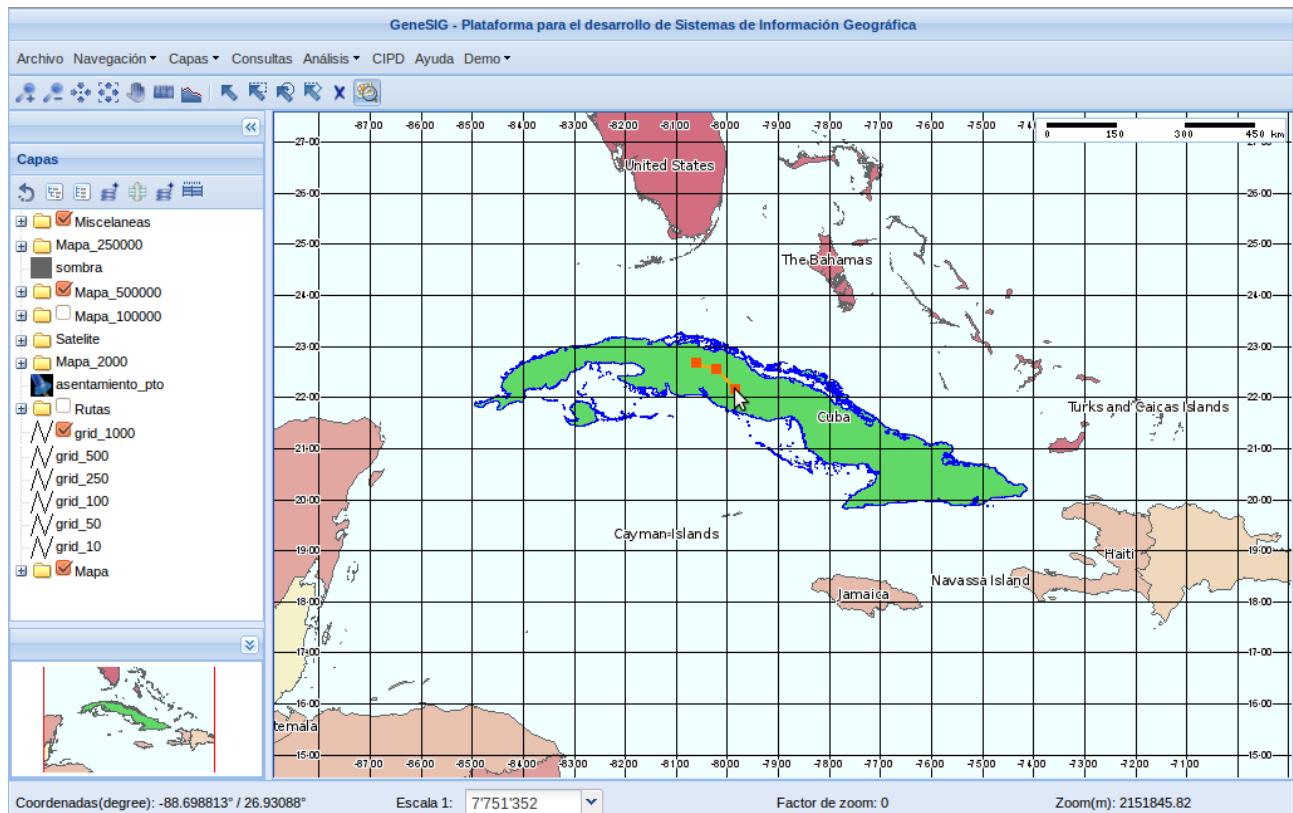


Ilustración 61 Emplear la herramienta de trazado del plugin Search

En función de lograr esto a través del método *resetMapEventHandlers* se procede a resetear todo los manejadores de eventos sobre el mapa definidos por el API cliente de GeneSIG, evitando que se quede colgado alguno anteriormente activado y se produzcan efectos no deseados. Posteriormente se procede a definir la capa donde se almacenara los elementos grafico que componen el trazado, especificándole un identificador semántico de forma que pueda ser controlado en otro momento, para ello se emplea la función *setCurrentLayer*.

Finalmente se activa el recurso de dibujado para trazado lineal a través de la función *setTool* especificándole el parámetro '*draw.line*' , la cual procede de un objeto generado por el nucleo del API de la plataforma al cual es posible acceder por mediación de la llamada a la función *this.getDisplay(aDisplay)*.

Una vez activada la herramienta se hace necesario definir los eventos necesarios que permitirán almacenar la selección de puntos pertenecientes al trazo generado, los cuales se caracterizan por llevar como prefijo el termino *on*, a continuación se procede a describir cada uno de estos.

La función *onClic* se activa una vez accionado clic izquierdo de *mouse*, la cual recibe por parámetro una estructura con los datos de las coordenadas seleccionadas. De forma

similar *onMove* permite obtener las coordenadas geográficas asociadas al puntero del ratón basada en el mínimo movimiento de dicho dispositivo externo.

Una vez iniciado el uso de la herramienta es necesario limpiar posibles trazos creados anteriormente en función de ganar en cuanto a claridad sobre los datos presentados en pantalla. Esto es posible a través de la función denominada *onNewFeature*.

Por otra parte el comportamiento asociado al evento generado por la acción de doble clic izquierdo de mouse es considerado clave en este módulo pues identifica el fin de la selección y es donde se procede a almacenar los datos para ser enviado al servidor. Esto se logra a través de la función *onFeatureInput*, la cual recibe por parámetro una estructura que posee una propiedad denominada vértices, donde se almacena el listado de coordenadas que componen dicha selección.

En este caso se procede a incorporar al listado de coordenadas, la distancia cumulada entre los puntos geográficamente distanciados, determinándose a través de la clase para el manejo de cálculos genéricos sobre geometrías espaciales denominada *GenericGeometryCalculation*. Una vez concluido dicho proceso se emplea el recurso *CartoWeb.triggerLight* especificadole la acción *Perfil.Perform* para enviar los datos al servidor.

Por último los eventos *onToolUnset* y *onCancel* permiten controlar las señales emitidas una vez que se desactiva la herramienta o se cancele su utilización.

Código perteneciente al fichero search.ajax.js, responsable de la comunicación con el servidor a través de Ajax:

```
AjaxPlugins.Perfil = {
    handleResponse: function(pluginOutput)
    {
        if(AjaxPlugins.Perfil.Aux.HPerfil)
        {
            AjaxPlugins.Perfil.Aux.Generar_Grafica(pluginOutput);
        }
    },
    init: function(){
        objView.addToolBarButton({
            text: "",
            icon:"perfil",
            ttip:"Cálculo del Perfil Sobre el Terreno",
            id: 'perfil',
            handler: AjaxPlugins.Perfil.handler.calculate
        });
        Ext.getCmp('perfilmenu').on('click', AjaxPlugins.Perfil.handler.calculate);
    },
    Actions: {
        Perform: {
            buildPostRequest: function(argObject)
            {
                AjaxPlugins.Perfil.Aux.HPerfil = true;
                if(AjaxPlugins.Perfil.GUI)
                    AjaxPlugins.Perfil.GUI.ShowPerfilWindow('Información de Perfiles');

                var value = AjaxPlugins.Perfil.GUI.ComboBox1.getValue();
                if(AjaxPlugins.Perfil.GUI.ComboBox1)
                    AjaxPlugins.Perfil.Aux.Factor = parseInt(value);
                if(!AjaxPlugins.Perfil.Aux.Factor) AjaxPlugins.Perfil.Aux.Factor = 900;

                return AjaxHelper.ToQueryString("ListCoord", AjaxPlugins.Perfil.Aux.Coord)
                    + AjaxHelper.ToQueryString("Factor", AjaxPlugins.Perfil.Aux.Factor);
            }
        },
        Send: {
            buildPostRequest: function(argObject)
            {
```

```

        return AjaxHelper.ToQueryString("ListCoord", AjaxPlugins.Perfil.Aux.Coord));
    },
    onAfterAjaxCall: function(argObject, pluginOutput){
        var obj = Ext.decode( pluginOutput.htmlCode['data'] );
        var info = pluginOutput.variables.count;
        alert( obj[val] + " : " + info );
    }
},
handler: {
    calculate : function()
    {
        if( Map.prototype.perfil )
            Genesig.Tools.enableTool('perfil');
        else
        {
            loadDynamicConfig.executeLoadPlugin("perfil", function(){
                Genesig.Tools.enableTool('perfil');
            });
        }
    }
},
AjaxPlugins.initializablePlugins.push(AjaxPlugins.Perfil);

```

Obsérvese como en la función *init* se construyen tanto el botón que aparece en la barra de herramientas, como la opción del menú contextual, en este caso como el comportamiento es el mismo se define con el nombre de *calculate* dentro del objeto denominado *handler*. Otro importante a destacar es que la implementación se basa en la carga dinámica de los *js*, por tanto se emplea el recurso *loadDynamicConfig.executeLoadPlugin* al cual se le deben especificar por parámetro: el nombre del plugin, así como el evento que se desee ejecutar una vez culminado el proceso de carga, permitiéndole ser sincronizado con la propia acción .

El método clave en este ejemplo es *buildPostRequest* perteneciente al *ajaxaction* denominada *Perform*, en el cual se construye una ventana permitiendo recoger los datos necesarios que componen la petición que será realizada al servidor, tal y como se muestra en la imagen que aparece a continuación.

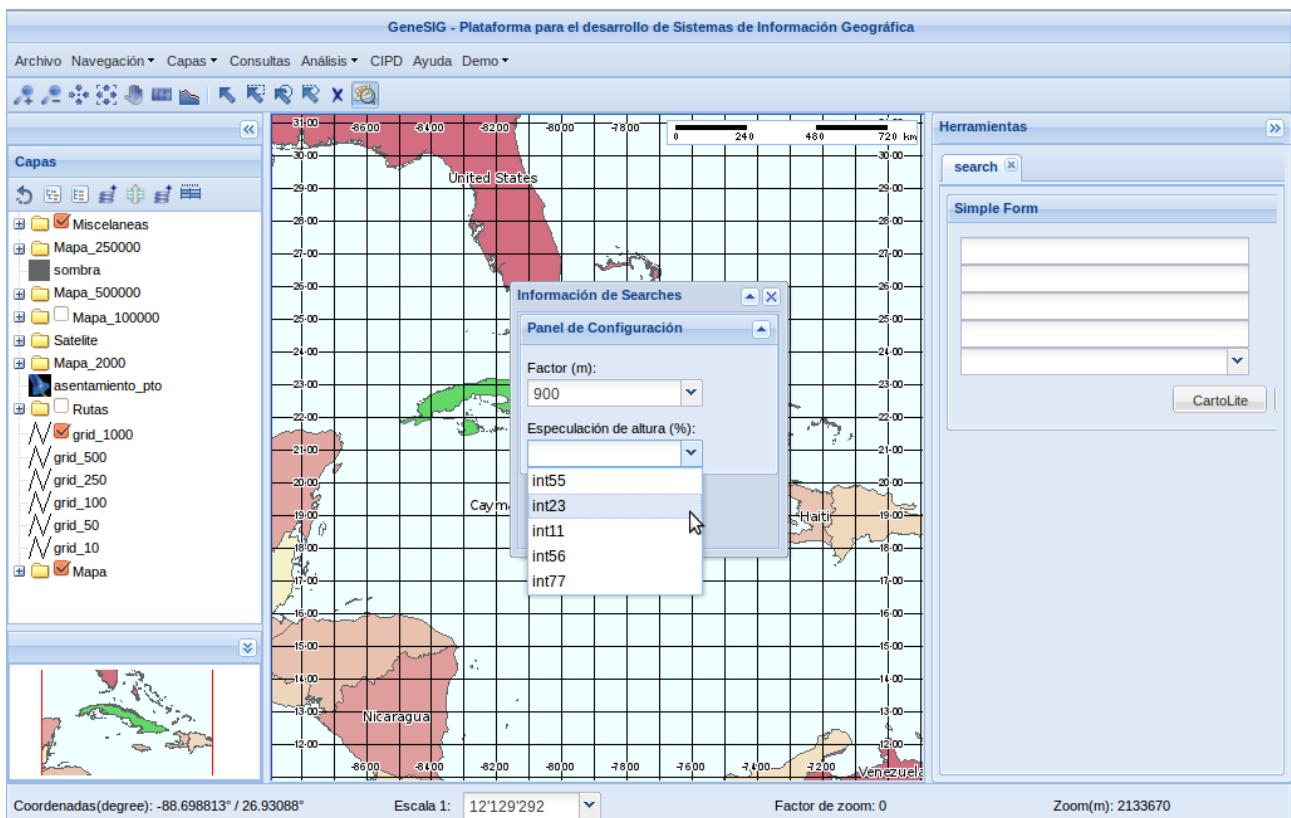


Ilustración 62 Especificar los datos del formulario para el plugin Search

Por otra parte el hecho de implementarse el método *handleResponse*, implica que sin importar que tipo de petición fue realizada este debe adoptar un comportamiento determinado. Sin embargo al implementar la función *onAfterAjaxCall* definida para el *ajaxaction* denominada *Send*, limita su respuesta únicamente para las invocaciones asociadas a dicha acción, que en este caso consiste en mostrar un mensaje de alerta con la salida generada por el servidor en determinado formato.

En este ejemplo se implementaron otros ficheros como es el caso de *search.aux.js* y *search.gui.js*. En el primero se incorporaron funciones de carácter auxiliar en función de complementar el comportamiento del plugin, así como garantizar algunas validaciones y en el segundo se incorporó todos los elementos relacionados con la interfaz gráfica de usuario.

Otro elemento que se hace imprescindible mencionar sobre todo para el desarrollo de esta fase es el empleo del *Firebug*, una extensión de Firefox muy útil para desarrolladores del web, que permite examinar minuciosamente cada uno de los elementos de la página, en busca de errores del código o fallos de presentación. Uno de los recursos más relevantes del mismo es la posibilidad de darle seguimiento a la ejecución del código java script en tiempo real, especificando los puntos de ruptura podemos controlar los valores que toman las variables en cada instante, con tan solo pasar el mouse por encima de estas o especificarle la opción de agregar seguimiento para evaluarlas en la región ubicada a la derecha, tal y como se muestra en la imagen que aparece a continuación.

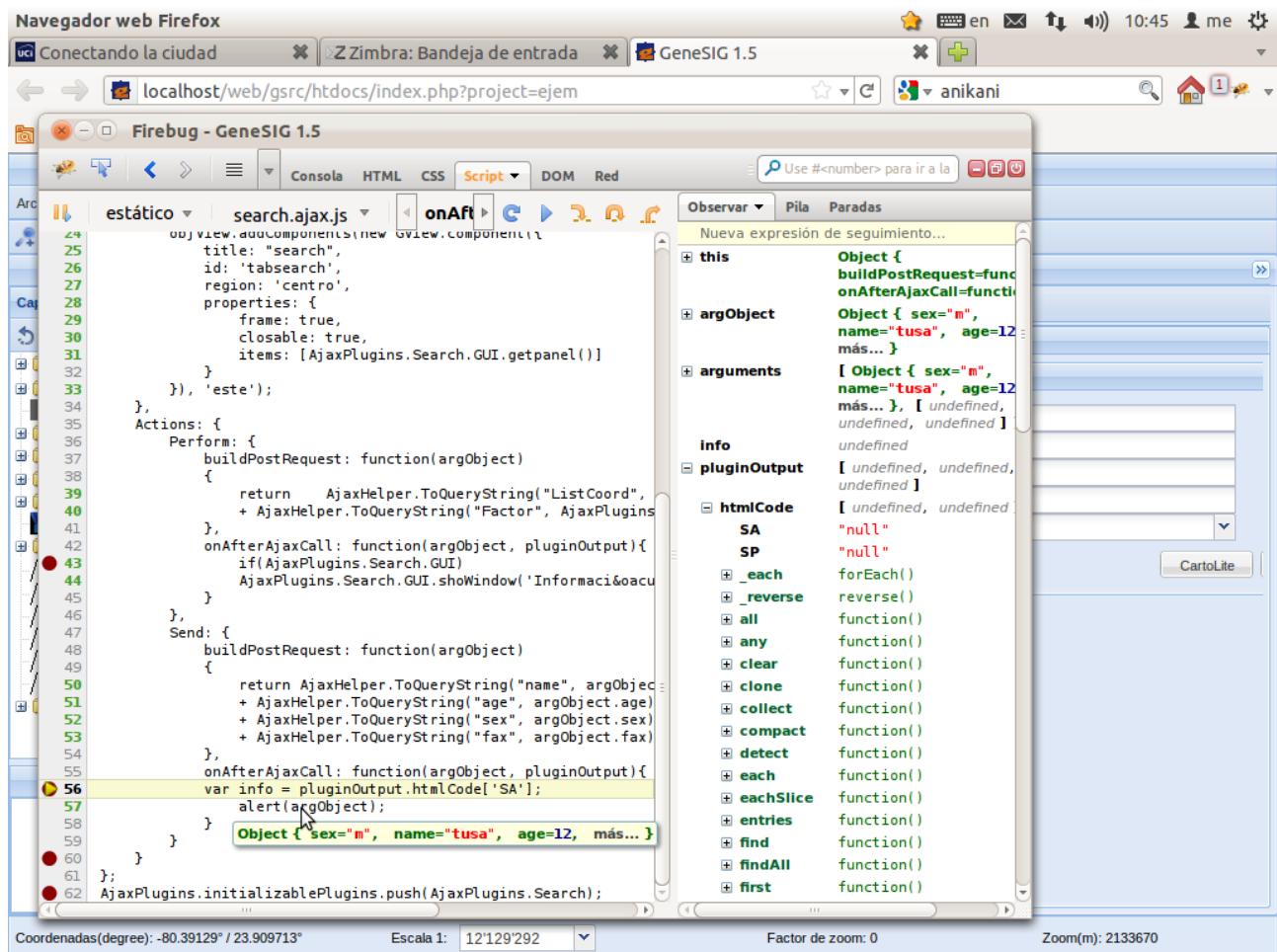


Ilustración 63 Traseando el código cliente del plugin Search

Tambien es posible inspeccionar las respuestas emitidas por el servidor, permitiendo obtener la información básica asociada a los mismo tales como tiempo de respuesta, método de envío, dominio, ip remota, parámetros, encabezados, entre otros, para mejor comprensión refiérase a la figura que se muestra a continuación.

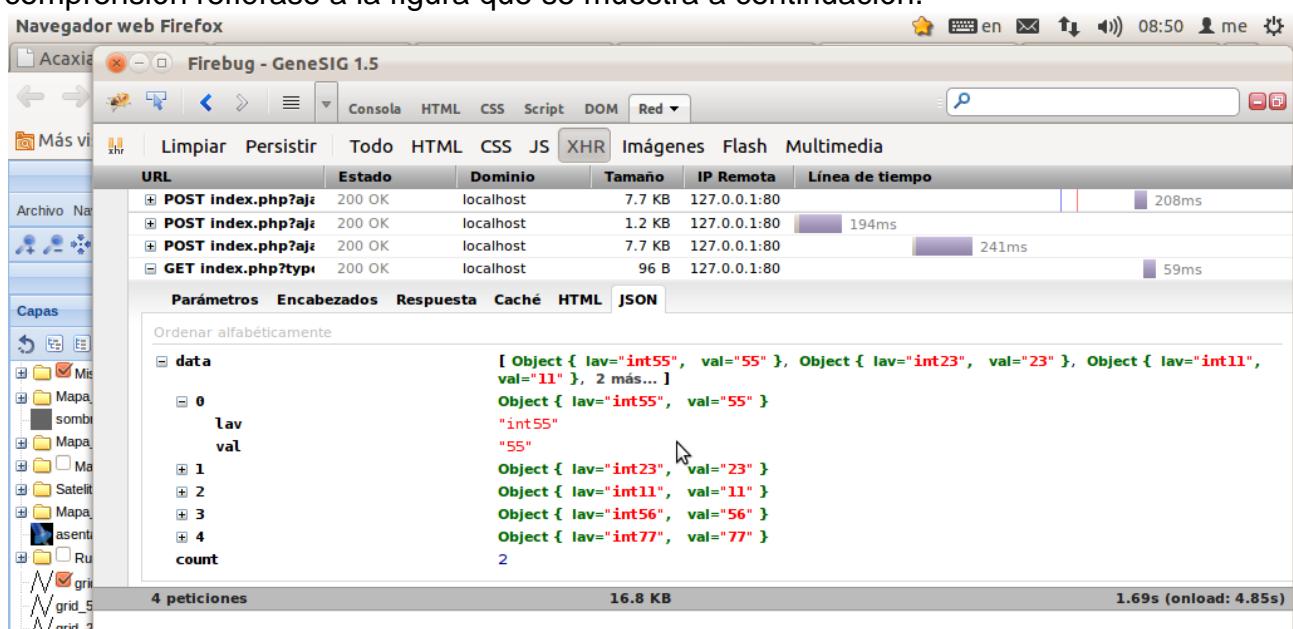


Ilustración 64 Observar la respuesta del plugin Search para el formato json

Como peculiaridad de la plataforma la gran mayoría de las transacciones realizadas entre

cliente y servidor se hacen mediante el formato xml. A través de este recurso es posible evaluar el resultado permitiendo identificar posibles errores que de lo contrario se convertiría en un proceso realmente tedioso. Además también es posible evaluar el rendimiento de nuestras peticiones para de esta seleccionar el mecanismo de envío más adecuado a nuestras necesidades.

URL	Estado	Dominio	Tamaño	IP Remota	Línea de tiempo
POST index.php?ajx	200 OK	localhost	7.7 KB	127.0.0.1:80	210ms
POST index.php?ajx	200 OK	localhost	118 B	127.0.0.1:80	239ms
POST index.php?ajx	200 OK	localhost	1.1 KB	127.0.0.1:80	261ms
POST index.php?ajx	200 OK	localhost	7.7 KB	127.0.0.1:80	263ms
GET index.php?type	200 OK	localhost	236 B	127.0.0.1:80	48ms

Parámetros Encabezados Respuesta Caché XML

```
<pluginResponse>
    <plugin name="search">
        <htmlCode id="data" value='[{"lav": "int55", "val": "55"}, {"lav": "int23", "val": "23"}, {"lav": "int11", "val": "11"}, {"lav": "int56", "val": "56"}, {"lav": "int77", "val": "77"}]'></htmlCode>
        <htmlCode id="count" value="2"></htmlCode>
    </plugin>
    <plugin name="cartoMessages">
        <variable id="userMessages" value="[]"/>
        <variable id="developerMessages" value="[]"/>
    </plugin>
</pluginResponse>
```

5 peticiones 16.9 KB 2.62s (onload: 6.39s)

Ilustración 65 Observar la respuesta del plugin Search para el formato xml

Una vez concluido el desarrollo de los elementos que componen para parte cliente del modulos, podemos iniciar la segunda fase.

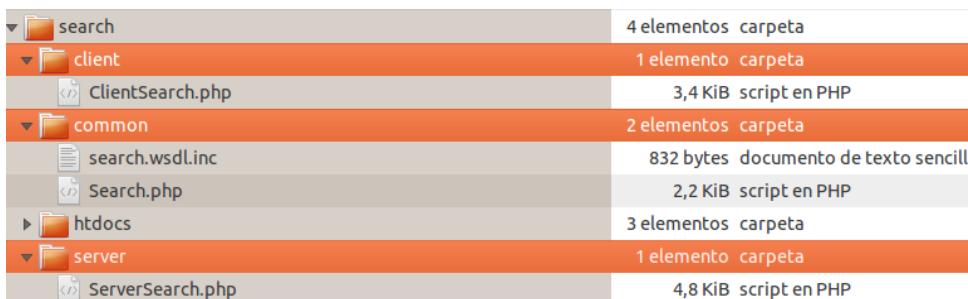


Ilustración 66 Estructura de la parte servidora del plugin Search

El objetivo de esta parte servidora es bastante simple, a partir de un listado de puntos geográficamente referenciados se obtendrán los valores de altura que les corresponden y una vez estructurada dicha información se arrojarán al cliente para que la consuma. De esta forma se estarían poniendo en práctica algunos mecanismos provistos por GeneSIG en función de garantizar la interacción con el servidor de mapas.

Contenido del fichero *ClientSearch.php*, cuya responsabilidad consiste en el tratamiento de los datos provenientes de la parte cliente del plugin.

```
<?php
class ClientPerfil extends ClientPlugin implements GuiProvider, ServerCaller, Ajaxable
{
    //----- Declaración de variables -----
    private $listCoord;
    private $serialArea;
    private $serialPunto;
```

```

private $factor;
//----- Declaración de funciones -----
public function __construct()
{
    parent::__construct();
    $this->listCoord = array();
    $this->factor = 900;
}
//----- Comunicación con el Smarty -----
protected function renderFormPrepare() {}
public function renderForm(Smarty $template) {}
//----- Comunicación con el cliente -----
public function handleHttpPostRequest($request)
{
    if (isset($request['ListCoord'])) 
    {
        $listJson = $request['ListCoord'];
        $this->factor = $request['Factor'];
        $this->listCoord = json_decode(stripslashes($listJson), false);
    }
}
//----- Comunicación con el cliente -----
public function handleHttpLightRequest($request)
{
    $this->handleHttpPostRequest($request);
}
public function handleHttpGetRequest($request)
{
    $this->handleHttpPostRequest($request);
}
public function ajaxGetPluginResponse(AjaxPluginResponse $ajaxPluginResponse)
{
    $serialPunto = json_encode($this->serialPunto);
    $serialArea = json_encode($this->serialArea);
    $ajaxPluginResponse->addHtmlCode('SP', $serialPunto);
    $ajaxPluginResponse->addHtmlCode('SA', $serialArea);
}
public function ajaxHandleAction($actionName, PluginEnabler $pluginsDirectives)
{
    switch ($actionName) {
        case 'Search.Perform':
            $pluginsDirectives->disableCoreplugins();
            $pluginsDirectives->enablePlugin('search');
        break;
        default $pluginsDirectives->disablePlugin('search'); break;
    }
}
//----- Comunicación con el servidor -----
public function buildRequest()
{
    $T = count($this->listCoord);
    if ($T)
    {
        $perfilRequest = new PerfilRequest();
        $perfilRequest->factor = $this->factor;
        $perfilRequest->coordenadas = $this->listCoord;
        return $perfilRequest;
    }
    else return 0;
}
public function initializeResult($perfilResult)
{
    if (!empty($perfilResult))
    {
        $this->serialPunto = $perfilResult->serialPunto;
        $this->serialArea = $perfilResult->serialArea;
    }
}
public function handleResult($perfilResult) {}

```

```
}
```

En este caso el procesamiento de los datos provenientes de la parte cliente es mínimo, teniendo en cuenta que la complejidad de este ejemplo es baja, tan solo se procede a decodificar del formato json y almacenarlos en variables globales de la clase, en función de que puedan ser leídas durante la ejecución del plugin. De esta forma se construye la petición al servidor empleando la clase *SearchRequest* y se captura su respuesta a partir de una instancia generada de *SearchResult*, generándose una salida hacia el navegador, la cual consiste de dos arreglos previamente codificados al formato json. Para mayor comprensión refiérase a los acápitulos 5.5.2 *Comunicación Client.js con Client.php* y 5.5.3 *Comunicación Client.php con Server.php*

Contenido del fichero *Search.php*:

```
<?php
    require_once(CARTOWEB_HOME . 'common/CwSerializable.php');
    //----- Clase para controlar las peticiones del cliente al servidor -----
    class SearchRequest extends CwSerializable
    {
        //----- Declaración de variables -----
        public $factor;
        public $coordenadas;
        //----- Declaración de funciones -----
        public function unserialize($struct)
        {
            $this->factor = self::unserializeValue($struct, 'factor', 'int');
            $this->coordenadas = self::unserializeArray($struct, 'coordenadas');
        }
    }
    //----- Clase para controlar las respuestas del servidor al cliente-----
    class SearchResult extends CwSerializable
    {
        //----- Declaración de variables -----
        public $serialArea;
        public $serialPunto;
        //----- Declaración de funciones -----
        public function unserialize($struct)
        {
            $this->serialArea = self::unserializeArray($struct, 'serialArea');
            $this->serialPunto = self::unserializeArray($struct, 'serialPunto');
        }
    }
?>
```

La petición que se le genera al servidor definida por la clase *SearchRequest*, consta de dos variables elementales factor y coordenadas. Las coordenadas son un listado ordenado que se componen por tres valores: latitud, longitud y distancia acumulada, los cuales se obtienen a partir de selección que realiza el usuario una vez culminado el trazado de puntos sobre el mapa. De igual forma el factor es un valor numérico que se especifica a través de un formulario una vez concluido el trazado del usuario. Por otra parte la respuesta se compone por dos variables, las cuales corresponden a una serie de puntos lineales en correlación al trazado especificado, así como una serie de putos que permiten describir un área.

Contenido del fichero *search.wsdl.inc*:

```
<!-- search wsdl -->
<complexType name="SearchRequest">
    <all>
        <element name="className" type="xsd:string"/>
        <element name="Factor" type="xsd:int"/>
        <element name="Coordenadas" type="types:ArrayOfCoord"/>
    </all>
</complexType>
```

```

<complexType name="SearchResult">
    <all>
        <element name="className" type="xsd:string"/>
        <element name="SerialArea" type="types:ArrayOfCoord"/>
        <element name="SerialPunto" type="types:ArrayOfCoord"/>
    </all>
</complexType>

<complexType name="ArrayOfCoord">
    <complexContent>
        <restriction base="enc11:Array">
            <attribute ref="enc11:arrayType" wsdl:arrayType="xsd:string[]"/>
        </restriction>
    </complexContent>
</complexType>
<!-- fin de Search -->

```

El fichero *wsdl* puede resultar innecesario en caso de que no se determine utilizar el modo de comunicación SOAP o se emplee en lenguajes débilmente tipado como el *php* que no lo requiera para efectuar la invocación. Sin embargo de ser así estarían limitando sus soluciones, pues no podrían ser utilizadas desde otras tecnologías o plataformas, elemento que tiene implicación directa en la integración con sistemas externos, así como la puesta en práctica del principio de no duplicidad de esfuerzos.

Contenido del fichero *ServerSearch.php*:

```

<?php
class ServerPerfil extends ClientResponderAdapter
{
    //----- Declaracion de Variables -----
    private $serieP;
    private $serieA;
    private $coordInter;
    private $msLayer;
    //----- Declaracion de Funciones -----
    public function __construct()
    {
        parent::__construct();
        $this->serieP = array();
        $this->serieA = array();
        $this->coordInter = array();
        $this->msLayer = false;
    }
    //----- Comunicacion con el Cliente -----
    public function handlePreDrawing($requ)
    {
        if($requ)
        {
            if($this->getQueryLayer())
            {
                $LD = 0;
                $FD = $requ->factor;
                $this->coordInter[0] = $requ->coordenadas[0];
                //--- Recepcion de datos del cliente php -----
                $L = count($requ->coordenadas);
                for($i=0; $i<$L-1; $i++)
                {
                    $DS = $requ->coordenadas[$i+1][2]-$requ->coordenadas[$i][2];
                    if($DS < $FD) $MD = 0;
                    else
                    {
                        $MD = round($DS/$FD);
                        $X = $requ->coordenadas[$i][0];
                        $Y = $requ->coordenadas[$i][1];
                        $DX = ($requ->coordenadas[$i+1][0] - $X)/$MD;
                        $DY = ($requ->coordenadas[$i+1][1] - $Y)/$MD;
                    }
                }
            }
        }
    }
}

```

```

        }

    for($j=0; $j<$MD; $j++)
    {
        $this->coordInter[$LD+1][0] = $DX + $this->coordInter[$LD][0];
        $this->coordInter[$LD+1][1] = $DY + $this->coordInter[$LD][1];
        $NX = $this->coordInter[$LD][0];
        $NY = $this->coordInter[$LD][1];

        if($NX!=$X && $NY!=$Y)
        {
            $C = count($this->SerieA);
            $this->cerieA[$C][0] = $LD * $FD;
            $this->cerieA[$C][1] = $this->queryByPoint($NX,$NY);
            $this->cerieA[$C][2] = $NX;
            $this->cerieA[$C][3] = $NY;
            }$LD++;
        }
    }

//--- Envio de datos al cliente php -----
$searchResult = new SearchResult();
$searchResult ->serialPunto = $this->serieP;
$searchResult ->serialArea = $this->serieA;
return $searchResult;
} else
{
    $this->serverContext->addMessage(
        $this, 1,
        "La capa << {$this->getConfig()->layerQuery} >> no existe en el Mapfile."
    );
}
}

public function queryByPoint($X, $Y)
{
    ms_ResetErrorList();

    $altura = 0;
    $msPoint = ms_newPointObj();
    $msPoint->setXY($X, $Y);

    if($msLayer = $this->getQueryLayer())
    {
        $status = $msLayer->status;
        $maxscaledenom = $msLayer->maxscaledenom;
        $minscaledenom = $msLayer->minscaledenom;
        $msLayer->set('maxscaledenom', -1);
        $msLayer->set('minscaledenom', -1);
        $msLayer->set('status', MS_ON);

        $qresult = @$msLayer->queryByPoint($msPoint, MS_SINGLE, -1);

        if ($qresult == MS_SUCCESS)
        {
            $msLayer->open();
            $nresult = $msLayer->getNumResults();
            if ($nresult)
            {
                $result = $msLayer->getResult(0);
                $shape = $msLayer->getShape(
                    $result->tileindex,
                    $result->shapeindex
                );
                $altura = $shape->values["value_0"];
                $shape->free();
            }
            $msLayer->close();
        }
        ms_ResetErrorList();
        $msLayer->set('maxscaledenom', $maxscaledenom);
        $msLayer->set('minscaledenom', $minscaledenom);
    }
}

```

```

        $msLayer->set('status', $status);
        return $altura;
    }

private function getQueryLayer()
{
    $msMapObj = $this->serverContext->getMapObj();
    if (!$this->msLayer)
        if ($this->msLayer = $msMapObj->getLayerByName($this->getConfig()->layerQuery))
            return $this->msLayer;
        else return false;
    else return $this->msLayer;
}

?>

```

Como bien se puede observar en la función *handlePreDrawing* se desarrolla la lógica relacionada con la obtención de las series numéricas. Básicamente se recorre el listado de puntos referenciados geográficamente, intentándose obtener nuevos valores por cada dos consecutivos, utilizándose para ello cálculos muy básicos, aunque esto podría mejorarse utilizando algunos métodos matemáticos donde el margen de error sea mínimo como la interpolación, entre otros, pero ese no es el objetivo de este material.

Por otra parte se implementó la función *queryByPoint*, la cual recibe por parámetro dos valores asociados a la latitud y longitud de un punto determinado. Esto permite obtener el valor de altura asociado al mismo que se encuentra almacenado en el MDE²⁸. En función de lograr esto se emplean los mecanismos establecidos por la librería Mapscript. Para garantizar un resultado fiable se procede a resetear la lista de errores del Mapserver a través de la función *ms_ResetErrorList*, se construye un objeto de tipo *msPointObj* el cual sería tomado como referencia de búsqueda, se solicita la instancia de la capa definida por su denominación semántica, almacenándose los valores de escala máxima, mínima y estado en variables temporales para luego poder restablecerlas una vez concluido el proceso, partiendo de la premisa que una layer que se encuentre apagada no puede ser consultada entre otros detalles.

Una vez que los datos necesarios estaban listos se procede a utilizar la función *queryByPoint* implícito en la clase *msLayerObj* la cual permite obtener una estructura de datos asociado al punto especificado, permitiendo ser accedidas por el método *getShape*.

5.11 Servicios

Uno de los mecanismos de integración que se están potenciando en la plataforma GeneSIG es el basado en Servicio Web. Este podría interactuar con servicios remotos a través de los métodos *get* y *post* de HTTP, pero SOAP es mucho más robusto y flexible. Permitiendo el paso de parámetros y comandos entre clientes y servidores de HTTP, independientemente de las plataformas y aplicaciones existentes entre ambas partes.

Los parámetros y los comandos se codifican utilizando XML.

SOAP²⁹ es un protocolo liviano, basado en lenguaje XML, para el intercambio de información estructurada en un ambiente descentralizado y distribuido. Sin embargo no define la aplicación, ni la semántica de implementación. En vez de esto, proporciona un modelo de empaquetamiento modular y los mecanismos para la codificación de los datos

²⁸ MDE acrónimo de Modelo Digital de Elevaciones

²⁹ SOAP acrónimo de Simple Object Access Protocol

dentro de los módulos. Esto permite que el protocolo se utilice en una amplia variedad de sistemas modulares, cumpliendo su propósito primario de facilitar la interoperabilidad entre componentes de software heterogéneos. En otros términos un paquete SOAP contiene información que se puede utilizar para invocar un método, sin embargo no define la forma de invocarlo y tampoco los mecanismos de recolección de basura distribuida, ni la seguridad de tipos o HTTP bidireccional.

5.11.1 Modo de acceso

Como bien se había descrito en la introducción del acápite 5. GeneSIG y en el 5.2 *Configuración*, es posible emplear el modo SOAP para la comunicación de los componentes internos, principalmente si estos se encuentran en un ambiente distribuido. En función de garantizar la interoperabilidad con sistemas externos, se toma como bases el mecanismo de comunicación anteriormente mencionado, permitiéndoles consumir la salida generada por la parte servidora de los plugin requeridos, para mayor comprensión refiérase a la figura que se muestra a continuación.

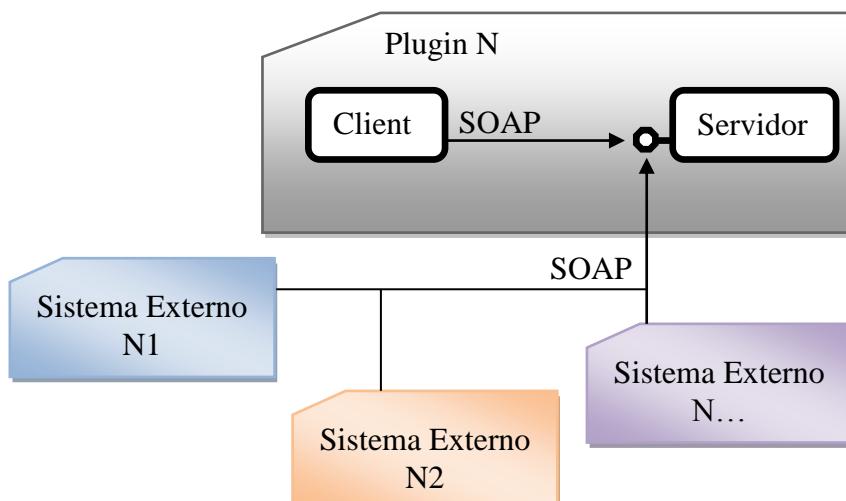


Ilustración 67 Mecanismo de comunicación en modo SOAP para un plugin

El consumo de estos servicios es tan simple como declarar una instancia de un cliente SOAP en dependencia de la librería y plataforma que se esté utilizado, especificándose la dirección del documento que describe el contrato de comunicación, el cual debe estar descrito basado en las especificaciones del formato wsdl, para mayor comprensión observe el ejemplo que se muestra a continuación.

```

<?php
$url = 'localhost/web/GeneSIG';
$mapid = 'ejem.geoweb';
$wsdl = "http://$url/htdocs/cartoserver.wsdl.php?mapId=$mapid";
$client = new SoapClient($wsdl, array());

```

Obsérvese que la dirección de dicho contrato se contralleva a partir de dos valores que generalmente pueden variar en función del entorno en que se encuentre desplegada la plataforma, estos son la *url* y el *mapid*. El primero consiste en la ruta de publicación de GeneSIG dentro del servidor web y el segundo consiste en un identificador semántico que se compone por la denominación del proyecto concatenado a través de un punto de la denominación del mapa, teniendo en cuenta que este producto puede estar compuesto por muchos proyectos y estos a su vez por varios mapas.

5.11.3 Elementos del contrato

Como bien se había descrito anteriormente el fichero generado basado en la especificación del wsdl, describe los tipos de datos que componen una transacción. Este elemento es sobre todo relevante para aquellas plataformas fuertemente tipadas como .Net, JRE, etc. A continuación se muestra un ejemplo de la estructura del mismo.

```
<complexType name="MapRequest">
  <all>
    <element name="mapId" type="xsd:string"/>
    <!-- ...elements specific to plugins... -->
  </all>
</complexType>
```

En función de garantizar una correcta comunicación, se procedió a definir una serie de estructuras las cuales serán utilizadas como tipos de datos nativos para la gran mayoría de las transacciones, tanto en la solicitud como en las respuestas las cuales se detallan a continuación.

En no pocas ocasiones se requiere utilizar un listado de cadenas de caracteres, a continuación se muestra la sintaxis de su especificación.

```
<complexType name="ArrayOfString">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="xsd:string[]"/>
    </restriction>
  </complexContent>
</complexType>
```

Concepto de dimensión, se define como una estructura compuesta por dos propiedades de tipo entera que definen el ancho y el alto del objeto respectivamente.

```
<complexType name="Dimension">
  <all>
    <element name="width" type="xsd:int"/>
    <element name="height" type="xsd:int"/>
  </all>
</complexType>
```

Concepto de cuadro delimitador también conocido como *bounding box*, se compone por cuatro valores numéricos de doble precisión, los cuales caracterizan los puntos extremos de dicho rectángulo.

```
<complexType name="Bbox">
  <all>
    <element name="minx" type="xsd:double"/>
    <element name="miny" type="xsd:double"/>
    <element name="maxx" type="xsd:double"/>
    <element name="maxy" type="xsd:double"/>
  </all>
</complexType>
```

La dimensión geográfica es una estructura que se compone por un objeto dimensión y otro que describe el cuadro delimitador.

```
<complexType name="GeoDimension">
  <all>
    <element name="dimension" type="types:Dimension"/>
    <element name="bbox" type="types:Bbox"/>
  </all>
</complexType>
```

```

</all>
</complexType>

```

Otro elemento muy utilizado es el punto, compuesto por dos valores numéricos de doble presicion.

```

<complexType name="Point">
  <all>
    <element name="x" type="xsd:double"/>
    <element name="y" type="xsd:double"/>
  </all>
</complexType>

```

Por tanto un listado de objetos puntos se puede definir como:

```

<complexType name="ArrayOfPoint">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Point[]"/>
    </restriction>
  </complexContent>
</complexType>

```

GeneSIG provee una estructura para describir figuras denominada shape, la cual se compone seis elementos numéricos de doble presicion y un listado de objetos puntos, tal y como se muestra a continuación.

```

<complexType name="Shape">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="x" type="xsd:double" minOccurs="0"/>
    <element name="y" type="xsd:double" minOccurs="0"/>
    <element name="minx" type="xsd:double" minOccurs="0"/>
    <element name="miny" type="xsd:double" minOccurs="0"/>
    <element name="maxx" type="xsd:double" minOccurs="0"/>
    <element name="maxy" type="xsd:double" minOccurs="0"/>
    <element name="points" type="types:ArrayOfPoint" minOccurs="0"/>
  </all>
</complexType>

```

De igual forma un listado de objetos shape sería:

```

<complexType name="ArrayOfShape">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Shape[]"/>
    </restriction>
  </complexContent>
</complexType>

```

5.11.3 Llamadas a procedimientos genéricos

La plataforma se encuentra provista por dos procedimientos genéricos *getMapInfo* y *getMap*, los cuales se encuentran públicos y descritos a través de un contrato definido en formato wsdl. Este último se construye a partir de una estructura central, así como la incorporación de aquellos ficheros con extensión *wsdl.inc* que se encuentra en el directorio *common* de cada uno de los plugin activos, de ahí la importancia de realizar una correcta definición de todas las estructuras de datos complejos que componen las clases de intercambio de datos entre cliente y servidor.

La funcionalidad *getMapInfo* permite obtener la información básica asociada al mapa

activo en la plataforma, permitiendo conocer de esta forma algunos elementos tales como el identificador semántico del mapa, su extensión territorial, dimensiones en pixels, listado de plugin cargados y por cada uno de estos mostraría los datos que los mismos sean capaces de publicar. Para mayor comprensión refiérase al ejemplo que se muestra a continuación.

```
<?php
try {
    $url = isset($_REQUEST['url']) ? $_REQUEST['url'] : 'localhost/web/GeneSIG';
    $mapid = isset($_REQUEST['mapid']) ? $_REQUEST['mapid'] : 'ejem.geoweb';

    $wsdl = "http://$url/htdocs/cartoserver.wsdl.php?mapId=$mapid&".time();
    $client = new SoapClient($wsdl, array());

    $result = $client->getMapInfo($mapid);

    echo "<pre>";
    print_r($result);

} catch (SoapFault $fault) {
    print $fault->faultstring;
}
?>
```

En este caso se decide que tanto la url base como el identificador de mapa debe ser especificador por parámetro en aras de ganar en genericidad, en caso de no ser especificados tomarían valores por defecto. Nótese que al construir la dirección donde se encuentra publicado el fichero wsdl se le concatena al final el valor generado por la función nativa de php denominada time, esto es útil en tiempo de desarrollo evitando que la generación del contrato sea almacenado en cache y de esta forma sea más rápido el proceso de gestión de cambios sobre el mismo. A continuación se muestra la salida en pantalla una vez ejecutado la función de impresión sobre el resultado arrojado.

```
stdClass Object
(
    [timestamp] => 1336628751
    [mapLabel] => GEOWEB_GIS
    [loadPlugins] => Array
        (
            [0] => distance
            [1] => perfil
            [2] => select
            [3] => search
        )
    [initialMapStates] => Array
        (
            [0] => stdClass Object
                (
                    [id] => default
                    [location] => stdClass Object
                        (
                            [bbox] => stdClass Object
                                (
                                    [minx] => -88.870862
                                    [miny] => 15.38674
                                    [maxx] => -68.962302
                                    [maxy] => 26.758788
                                    [className] => Bbox
                                )
                            [className] => InitialLocation
                        )
                    [layers] => Array()
                    [className] => InitialMapState
                )
        )
)
```

```
[extent] => stdClass Object
(
    [minx] => -88.870862
    [miny] => 15.38674
    [maxx] => -68.962302
    [maxy] => 26.758788
    [className] => Bbox
)
[location] =>
[keymapGeoDimension] => stdClass Object
(
    [dimension] => stdClass Object
    (
        [width] => 150
        [height] => 84
        [className] => Dimension
    )
    [bbox] => stdClass Object
    (
        [minx] => -88.870862
        [miny] => 15.38674
        [maxx] => -68.962302
        [maxy] => 26.758788
        [className] => Bbox
    )
    [className] => GeoDimension
)
[className] => MapInfo
[locationInit] => stdClass Object
(
    [scales] =>
    [minScale] =>
    [maxScale] =>
    [shortcuts] => Array()
    [recenterDefaultScale] =>
    [className] => LocationInit
    [fullExtent] => stdClass Object
    (
        [minx] => -88.870862
        [miny] => 15.38674
        [maxx] => -68.962302
        [maxy] => 26.758788
        [className] => Bbox
    )
)
```

La funcionalidad *getMap* es el recurso mas robusto y escalable de todo el andamiaje para la publicación de servicios provista por la plataforma. Esta a diferencia de otras tecnologías que se encargan de recrear todo un catalogo de servicios, permite centralizar todas las peticiones en una única llamada a un procedimiento remoto. Esto posibilita que los desarrolladores puedan incorporar, modificar o eliminar comportamientos que han sido publicados en un momento determinado sin que esto tenga una implicación directa sobre el resto de los sistemas que se definen como consumidores activos de GeneSIG.

Por tanto el consumo de servicios publicados por el marco de trabajo se realizara a través de la función pública denominada *getMap*. En cada petición se le debe especificar por parámetro un macro objeto de tipo *MapRequest*, el cual se define como contenedor global de todas las instancias de tipo *request* necesarias en función de los *plugin* que se deseen invocar.

Procedimiento para consumir las funcionalidades a través del servicio `getMap`

1. Declarar cada una de las clases request por cada una de las funcionalidades que se desean solicitar.
2. Declarar la clase MapRequest
3. Definir como propiedades publicas cada una de las funcionalidades que se desean solicitar.
4. Instanciar las propiedades definidas con los valores necesarios para obtener los resultados esperados.

Esto trae como ventaja que en una única solicitud se podría estar obteniendo información generada por diferentes módulos al mismo tiempo, permitiendo minimizar costos por concepto de rendimiento.

5.11.4 Mapa

El objeto de petición definido para el coreplugin *image* tiene una relación de composición con tres objetos de tipo *Image*, estos se denominan *mainmap* o mapa principal permitiendo realizar cualquier tipo de análisis sobre el mismo, *keymap* o mapa de referencia se define como una vista de mayor escala del área geográfica y por último pero no menos importante el objeto *scalebar*.

```
<?php
    class Image
    {
        public $className = "Image";
        public $isDrawn = "0";
        public $path = "";
        public $width = 500;
        public $height = 500;
        public $angle = 0;
        public $outputFormat = "";
    }

    class ImagesRequest
    {
        public $className;
        public $mainmap;
        public $keymap;
        public $scalebar;
        public $scalebarStatus = 0;

        public function __construct($width=500, $height=500, $idraw=true, $kdraw=false)
        {
            $this->className = "ImagesRequest";
            $this->scalebarStatus = 0;

            $this->mainmap = new Image();
            $this->mainmap->isDrawn = $idraw;
            $this->mainmap->width = $width;
            $this->mainmap->height = $height;
            $this->mainmap->angle = 0;

            $this->keymap = new Image();
            $this->keymap->isDrawn = $kdraw;
            $this->keymap->width = 100;
            $this->keymap->height = 100;

            $this->scalebar = new Image();
            $this->scalebar->isDrawn = $kdraw;
            $this->scalebar->width = 100;
            $this->scalebar->height = 100;
            $this->scalebar->format = "";
        }
    }
}
```

?>

Cada uno de estos objetos se componen por una serie de propiedades las cuales se enuncian a continuación:

- **isDrawn**: propiedad de tipo buleana que define el nivel de visibilidad de un objeto
- **path**: dirección física en la que será alojada la imagen generada del objeto
- **width**: propiedad numérica que expresa el valor del ancho que toma el objeto
- **height**: propiedad numérica que expresa el valor del alto que toma el objeto
- **angle**: propiedad numérica que expresa el valor del angulo de rotacion que toma el objeto
- **outputFormat**: propiedad que expresa el formato de salida

A continuación se muestra un ejemplo que pone en práctica los elementos anteriormente mencionados, permitiendo obtener la imagen del mapa principal asociado al usuario por defecto que toma la plataforma. En aras de minimizar la complejidad de dicho ejemplo se asume la declaración de las clases *Image* e *ImagesRequest* tal y como fueron descritas anteriormente.

```
<?php
try {
    class MapRequest
    {
        public $mapId;
        public $imagesRequest;

        public function __construct( $mapId )
        {
            $this->mapId = $mapId;
            $this->imagesRequest = new ImagesRequest();
        }
    }

    $url = isset ( $_REQUEST['url'] ) ? $_REQUEST['url'] : 'localhost/web/GeneSIG';
    $mapid = isset ( $_REQUEST['mapid'] ) ? $_REQUEST['mapid'] : 'ejem.geoweb';

    $wsdl = "http://$url/htdocs/cartoserver.wsdl.php?mapId=$mapid&". time();

    $soapCliente = new SoapClient($wsdl, array());
    $soapCliente->__setCookie('PHPSESSID', 'services-request');

    $mapRequest = new MapRequest( 'ejem.geoweb' );
    $mapResult = $soapCliente->getMap($mapRequest);

    header("location: http://$url/htdocs/{$mapResult->imagesResult->mainmap->path}");

} catch (SoapFault $e) {
    echo '<hr>' . $e->faultcode . '<br>' . $e->faultstring;
}
?>
```

Nótese que en este caso se puede entregar un resultado palpable con tan solo ejecutar el *script* anterior, empleándose para ello el recurso nativo de *php* que permite redireccionar la página hacia la *url* en que se encuentran alojadas las imágenes generadas. Siendo posible acceder a esta a través de los datos obtenidos en el *MapResult*, el cual consiste en una macro estructura similar al *MapRequest* pero más orientada a brindar información sobre la transacción. Para mayor comprensión del resultado obtenido véase la figura que se muestra a continuación.

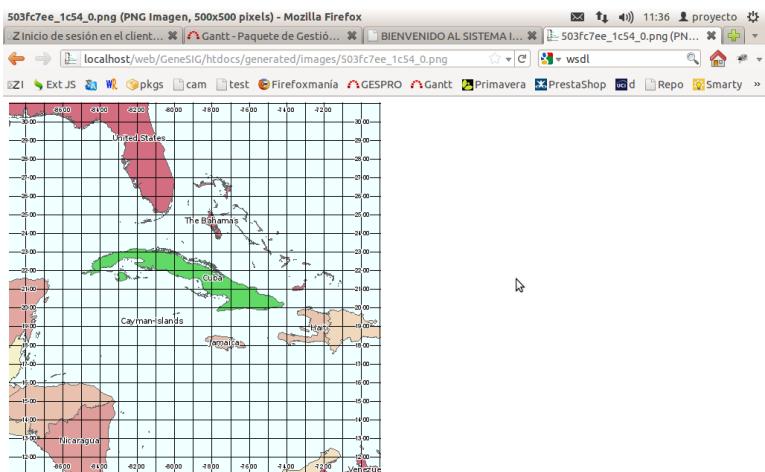


Ilustración 68 Resultado arrojado por el consumo de servicio de mapa

Como bien se puede observar tan solo es necesaria la información requerida por el *coreplugin* denominado *image*. Evidentemente se debe a la baja complejidad de este ejemplo, más adelante se irán mostrando fragmentos de código que ilustran una mayor complejidad en cuanto a interacción con el mapa se refiere.

5.11.5 Navegación

La navegación es uno de los elementos más relevantes en cuanto a interacción con mapas se refiere, teniendo en cuenta que mediante alguno de los conceptos que aquí intervienen es posible desplazarse espacialmente y visualizar el área que realmente se requiere con mayor o menor nivel de detalles. A continuación se muestra un fragmento de código que ilustra dicha afirmación.

```
<?php
    class LocationRequest
    {
        public $className;
        public $bboxLocationRequest;
        public $scale;
        public $locationType;

        public function __construct($minx=-88.870862,      $miny=15.386740,      $maxx=-68.962302,
$maxy=26.758788)
        {
            $this->className = 'LocationRequest';
            $this->locationType = 'bboxLocationRequest';
            $this->bboxLocationRequest->className = "BboxLocationRequest";
            $this->bboxLocationRequest->showRefMarks = true;
            $this->bboxLocationRequest->bbox->minx = $minx;
            $this->bboxLocationRequest->bbox->miny = $miny;
            $this->bboxLocationRequest->bbox->maxx = $maxx;
            $this->bboxLocationRequest->bbox->maxy = $maxy;
        }
    }

    class MapRequest
    {
        public $mapId;
        public $imagesRequest;
        public $locationRequest;

        public function __construct( $mapId )
        {
            $this->mapId = $mapId;
            $this->imagesRequest = new ImagesRequest();
            $this->locationRequest = new LocationRequest();
        }
    }
}
```

```

        }

    try {
        $url = isset($_REQUEST['url']) ? $_REQUEST['url'] : 'localhost/web/GeneSIG';
        $mapid = isset($_REQUEST['mapid']) ? $_REQUEST['mapid'] : 'ejem.geoweb';

        $wsdl = "http://$url/htdocs/cartoserver.wsdl.php?mapId=$mapid&".time();

        $soapCliente = new SoapClient($wsdl, array());
        $soapCliente->__setCookie('PHPSESSID', 'services-request');

        $mapRequest = new MapRequest('ejem.geoweb');
        $mapResult = $soapCliente->getMap($mapRequest);

        header("location: http://$url/htdocs/{$mapResult->imagesResult->mainmap->path}");

    } catch (SoapFault $e) {
        echo '<hr>' . $e->faultcode . '<br>' . $e->faultstring;
    }
}

?>

```

Observese que el objeto de tipo *LocationRequest* definido para el *coreplugin* denominado *location*, esta compuesto por una instancia del objeto *BboxLocationRequest*, el cual encapsula los pares de coordenadas expresados en [*longitud/latitud*] que corresponden a los puntos máximo y minimo que definen el rectángulo de visualización.

Otro elemento a tener en cuenta es el objeto de tipo *Request*, el cual define el comportamiento de dicho plugin, los posibles valores se describen a continuación:

- **bboxLocationRequest:** Permite recentrar el mapa a través de un rectángulos de visualización definido como *bounding box* o su contracción *bbox*
- **panLocationRequest:** Permite realizar desplazamiento geográficos tanto verticales como horizontales, a esta acción se le denomina con el término *panning*
- **zoomPointLocationRequest:** Permite recentrar el mapa a partir del punto especificado, incluyéndosele el valor de *zoom* relativo, así como el valor de la escala para ajustar dicha visualización
- **recenterLocationRequest:** Realiza el centrado del mapa a partir de los datos especificados en el mapfile

El restos de las propiedades deben ser especificadas en consecuencias con el tipo de objeto *Request* previamente descrito. Por ejemplo en caso de seleccionar *panLocationRequest* se hace necesario detallar la propiedad *PanDirectionType*, cuyo rango de valores de describe a continuación:

- **VERTICAL_PAN_NORTH:** Habilita el desplazamiento hacia el norte
- **VERTICAL_PAN_NONE:** Deshabilita el desplazamiento vertical
- **VERTICAL_PAN_SOUTH:** - Habilita el desplazamiento hacia el sur
- **HORIZONTAL_PAN_WEST:** Habilita el desplazamiento hacia el oeste
- **HORIZONTAL_PAN_NONE:** Deshabilita el desplazamiento horizontal
- **HORIZONTAL_PAN_EAST:** Habilita el desplazamiento hacia el este

Por otra parte en caso de seleccionarse *zoomPointLocationRequest* es necesario especificar el valor de la propiedad *zoomType*, tomando valores comprendidos entre:

- **ZOOM_DIRECTION_IN:** Permite realizar un acercamiento sobre el mapa, tomando como referencia una constante la cual toma por defecto valor 2

- **ZOOM_DIRECTION_NONE**: Se limita únicamente a realizar un recentrado del mapa tomando como referencia un punto
- **ZOOM_DIRECTION_OUT**: Permite realizar un alejamiento en el mapa, tomando como referencia una constante, la cual asume por defecto el valor 0.5
- **ZOOM_FACTOR**: Realiza el efecto de enfoque sobre el mapa a partir de un factor especificado.
- **ZOOM_SCALE**: Realizar un el efecto de enfoque sobre el mapa a partir del valor definido para la escala especificada

Para mayor comprensión sobre este tema observe el ejemplo que se muestra a continuación, en el cual se emplea la opción *zoomPointLocationRequest* especificándole el valor *ZOOM_SCALE* a la propiedad *ZoomType*.

```
<?php
    class LocationRequest
    {
        public $className;
        public $bboxLocationRequest;
        public $scale;
        public $locationType;

        public function __construct()
        {
            $this->className = 'LocationRequest';
            $this->locationType = 'zoomPointLocationRequest';
            $this->zoomPointLocationRequest->bbox->minx = -88.870862;
            $this->zoomPointLocationRequest->bbox->miny = 15.386740;
            $this->zoomPointLocationRequest->bbox->maxx = -68.962302;
            $this->zoomPointLocationRequest->bbox->maxy = 26.758788;
            $this->zoomPointLocationRequest->point->x = 550000;
            $this->zoomPointLocationRequest->point->y = 150000;
            $this->zoomPointLocationRequest->zoomType = 'ZOOM_SCALE';
            $this->zoomPointLocationRequest->scale = 200000;
        }
    }
?>
```

En este caso existen algunos atributos que solo deben ser especificados de acuerdo con la opción seleccionada. Tal es el caso de *zoomFactor*, la cual se emplea únicamente para cuando la propiedad *zoomType* toma valor *ZOOM_FACTOR*, así como *scale* para el valor *ZOOM_SCALE* de la misma.

5.11.6 Autenticación

El servicio de autenticación permite especificar durante una transacción que usuario está solicitando la misma, manteniéndose el control en todo momento de quienes intentan acceder a la información publicada y de esta forma garantizar los elementos básicos de seguridad, sobre todo para este tipo de aplicaciones.

```
class AuthRequest
{
    public $usuario;
    public $password ;
    public $id;
    public function __construct($user, $passwod, $id)
    {
        $this->usuario = $use;
        $this->contrasena = $passwod;
        $this->id = $id;
        $this->className = '';
    }
}
```

```

    }

$mapRequest = new MapRequest( 'ejem.geoweb' );
$mapRequest->authRequest= new AuthRequest(
    'Instalacion',
    'Instalacion',
    '1300100000002203'
);
$mapResult = $soapCliente->getMap($mapRequest);

```

En caso de no especificarse dicha propiedad como en los ejemplo anteriormente descritos, la plataforma asume por defecto que la cuenta de usuario es la de invitado. Este elemento es realmente importante, atendiendo a que GeneSIG está implementada para trabajar en función de perfiles cartográficos denominados mapfile y que se generan forma exclusiva para cada usuario.

5.11.7 Identificación

Los servicios de consulta permiten obtener todos aquellos elementos que se encuentran encapsulados en las la capas activas sobre el mapa y que se encuentren dentro de los límites definidos por la selección especificada. En función de llevar a cabo la ejecución del plugin denominado *identificacion* se emplea una instancia de tipo *IdentificacionSoapRequest*. Esta estructura se compone por una lista de identificadores semánticos de capas, así como un objeto de tipo Shape, en el cual se define el Bbox. A continuación se muestra un ejemplo que ilustra dicha afirmación.

```

<?php
    $map = new MapRequest( 'ejem.geoweb' );

    $map->identificacionRequest->className = 'IdentificacionSoapRequest';
    $map->identificacionRequest->layersArray = array('MapaMundo');

    $map->identificacionRequest->shape->className = 'Rectangle';
    $map->identificacionRequest->shape->minx = -88.870862;
    $map->identificacionRequest->shape->miny = 15.386740;
    $map->identificacionRequest->shape->maxx = -68.962302;
    $map->identificacionRequest->shape->maxy = 26.758788;

    $mapResult = $soapCliente->getMap($map);

    echo '<img src=' . 'http://localhost/web/GeneSIG/htdocs/' . $result->imagesResult->mainmap->path. ' >';
    echo '<pre>';

    print_r( $result->identificacionResult->result );
?>

```

Como bien se puede observar se le incorpora la propiedad *identificacionRequest* al objeto de tipo *MapRequest*. Esto es equivalente crear una instancia de tipo *IdentificacionSoapRequest* y asignársela a dicho objeto, sin embargo en este caso se aprovechan las bondades del lenguaje *php* en función de incorporar de forma dinámica un conjunto de propiedades, es por ello que en este tipo de declaración es tan importante especificar la propiedad *className*. Otro elemento interesante es que este plugin aplica un color amarillo al área identificada, para mayor comprensión véase la figura que se muestra a continuación.

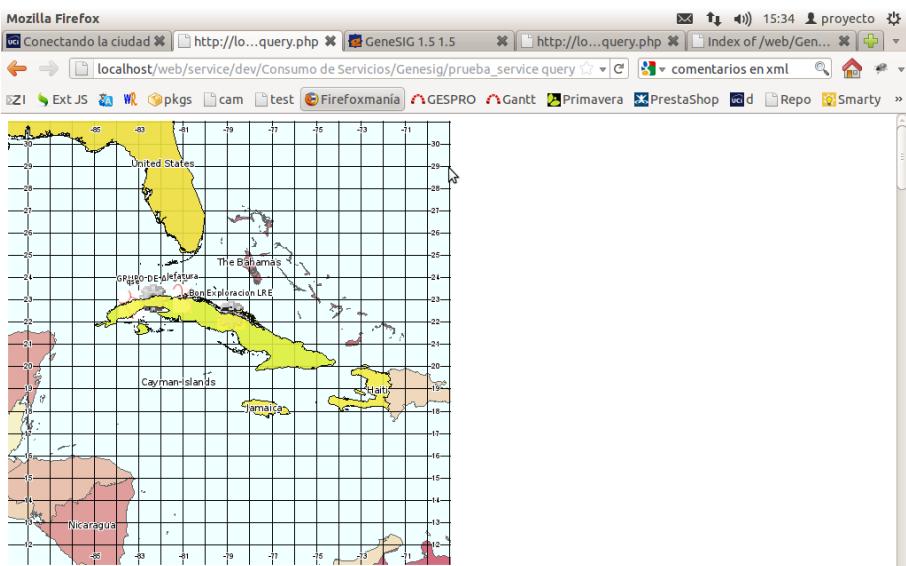


Ilustración 69 Vista resultante de plugin Identificacion

Como resultado de la ejecución de dicho plugin se obtiene un listado de estructuras de tipo Array que encapsula las propiedades de cada uno de los elementos encontrados, a continuación se expone una muestra como resultado de la impresión de los datos contenidos en `$result->identificacionResult->result`

```

Array
(
    [0] => Array
    (
        [index] => 23
        [type] => 2
        [classindex] => 0
        [numvalues] => 22
        [text] =>
        [bounds] => Array
        (
            [minx] => -89.223465
            [miny] => 15.889429
            [maxx] => -87.472511
            [maxy] => 18.496527
        )
        [numlines] => 43
        [tileindex] => 0
        [values] => Array
        (
            [gid] => 23
            [name] => Belize
            [iso_3166_2] => BZ
            [iso_3166_3] => BLZ
            [iso_3166_n] => 84
            [internet] => BZ
            [fips_10_4] => BH
            [com] =>
            [pop_2003] => 266440
            [pop_1990] => 190782
            [pop_2010] => 312260
            [births] => 30.46
            [deaths] => 6.05
            [migration] => 0.00
            [p_natgrowth] => 2.44
            [p_growthrate] => 2.44
            [inf_morb] => 27.07
            [inf_mortm] => 30.56
            [inf_mortf] => 23.42
            [lifeexp_b] => 67.36
            [life_expm] => 65.19
        )
    )
)

```

5.11.8 Tematización

Los mapas temáticos resultan de gran importancia debido a la facilidad que brindan para la representación de indicadores y análisis de datos. La cartografía temática recolecta y elabora tanto datos primarios cualitativos como cuantitativos, procesandolos con el fin de dar a conocer información de un tema o ciencia específicos (población, cobertura vegetal, catastro, aspectos culturales, aspectos económicos, entre otros) bajo una representación espacial a través de mapas gráficos, diagramas y perfiles. Ellos deben cumplir con el objetivo de evocar en la mente del lector una imagen precisa y clara del ambiente espacial del fenómeno.

Las características que se representan en los mapas temáticos como se había mencionado provienen de datos cualitativos y cuantitativos. En un mapa cualitativo se muestra la distribución espacial o la situación de un grupo de datos nominales, una de sus características es que en este tipo de mapa no se pueden determinar relaciones de cantidad. Los datos que representan estos mapas son puntuales, lineales y de superficie:

- **Mapas cualitativos de datos puntuales:** Definen la localización, la diferenciación y la naturaleza de la información. Los datos puntuales se representan generalmente por símbolos geométricos, pictóricos y letras.
- **Mapas cualitativos de datos lineales:** Definen al usuario representaciones de líneas diferentes, por ejemplo caminos, ríos, fronteras.
- **Mapas cualitativos de superficie:** Representan información sobre la distribución de las características cualitativas de una determinada área. Estos mapas son muy comunes en la representación zonal de los suelos, la geología, la vegetación, las áreas turísticas, y van acompañados de información descriptiva del tema.

El recurso que permite clasificar las diferentes peticiones en función de las necesidades de la tematización requerida es el *className*, la cual puede tomar valores comprendidos entre [*ThematicColorRequest* / *ThematicChartRequest* / *ThematicSymbolRequest*], permitiendo crear respectivamente mapas tematizados de tipo Corocromático y Coropleta, Gráficas Dinámicas (barra, pastel), así como de Símbolos Proporcionales.

A modo de precondición este servicio solo depende de la definición del *location* para los temas de ubicación, así como el *image* en función de definir el formato de salida, teniendo en cuenta que este servicio en si mismo implica un resultado visual. La autenticación es necesaria en función de que tipo de cartografía se desee emplear, así como los mecanismos de seguridad definidos para las misma en el ambiente en que se encuentre desplegado el sistema, por tanto es algo opcional en dependencia del entorno, este comportamiento se le aplica al resto de los servicios relacionados con autenticación.

A continuación se muestra un ejemplo que ilustra como puede generarse un mapa tematizado de tipo Gráficas Dinámicas. La idea de este recurso consiste en generar una capa nueva sobre el mapa, donde se asocien mediante un grafico de barra una serie de indicadores con un color que los identifique por individual.

```
<?php
$map = new MapRequest( 'ejem.geoweb' );
$map->mapaTematicoRequest->capa = "MapaMundo";
$map->mapaTematicoRequest->className = "ThematicChartRequest";
$map->mapaTematicoRequest->criterio = 3;
$map->mapaTematicoRequest->colorTematico = "#c2e7f0##68a8ce##4ac4ab##b580fe##315cf4";
$map->mapaTematicoRequest->valuesTematico = 'world.life_expf$world.migration $world.p_natgrowth
$world.inf_mortb $world.life_expm';
$mapResult = $soapCliente->getMap($map);
echo '<img src='.$http://localhost/web/GeneSIG/htdocs/'.$result->imagesResult->mainmap->path.' >';
?>
```

Nótese que el valor definido por la propiedad *criterio* define el tipo de gráfica a utilizar, en caso de especificarse con valor 3 genera un grafico de barra y si se especifica con valor 1 el resultado sería uno de pastel. Por otra parte la propiedad denominada *colorTematico* describe el rango de colores definido para cada atributo, dichos valores se especifican en forma de cadena y se concatenan a través del carácter *\$*, de igual forma pasa con la propiedad *valuesTematico*, que define el rango de propiedades, estas deben de coincidir con el orden definido para *colorTematico* para que puedan ser asociadas.

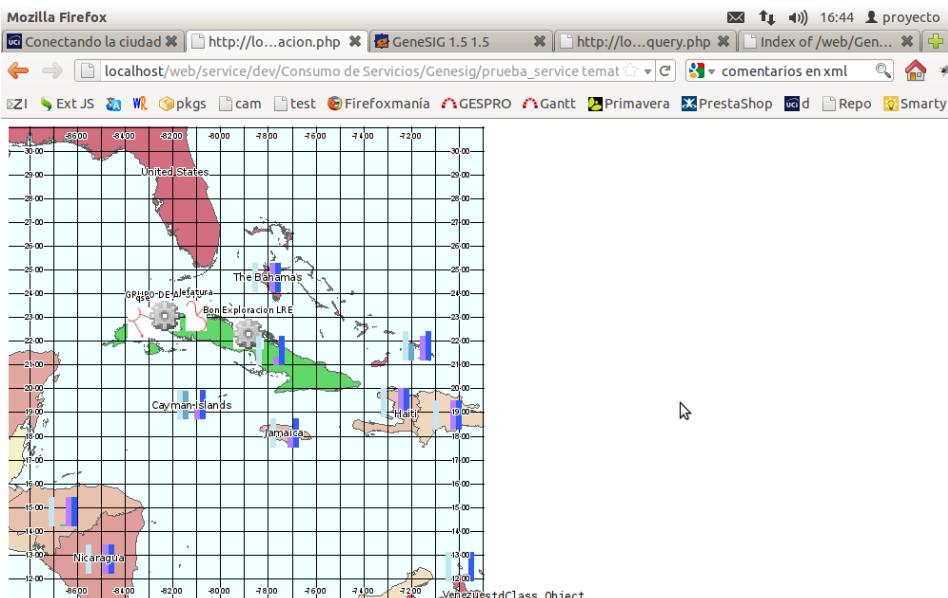


Ilustración 70 Tematización de tipo Gráficas Dinámicas

A continuación se muestra un ejemplo que ilustra como puede generarse un mapa tematizado de tipo Corocromático y Coropleta. En este caso se debe seleccionar un indicador numérico intrínseco de la capa especificada, para el cual se definirán un rango de posibles valores a los cuales se le deberá asignar un color determinado, permitiendo dibujar de un mismo color zonas atendiendo a dicho indicador.

```
<?php
$map = new MapRequest( 'ejem.geoweb' );

$map->mapaTematicoRequest->className = "ThematicColorRequest";
$map->mapaTematicoRequest->capa = "MapaMundo";
$map->mapaTematicoRequest->criterio = 'births';
$map->mapaTematicoRequest->operador = "numeric";
$map->mapaTematicoRequest->valuesTematico = '8.02$16.324$24.628$32.932$41.236$49.54';
$map->mapaTematicoRequest->colorTematico = "#001122#$9999ff#$ff66c0#$3333ab#$ff0096";

$mapResult = $soapCliente->getMap($map);
echo '<img src='.$http://localhost/web/GeneSIG/htdocs/'.$result->imagesResult->mainmap->path.' >';
?>
```

En este caso la propiedad denominada *criterio* describe el identificador semántico del indicador seleccionado en la capa que será tomada como base, se incorpora la propiedad *operador* especificando el valor *numérico*, el cual indica que se establecerá una relación numérica con los colores especificados entre las propiedades *valuesTematico* y *colorTematico*, a diferencia del caso anterior donde se asociaban campos a colores, aquí se relacionan rangos numéricos a colores. Para mayor comprensión véase la figura que se muestra a continuación.

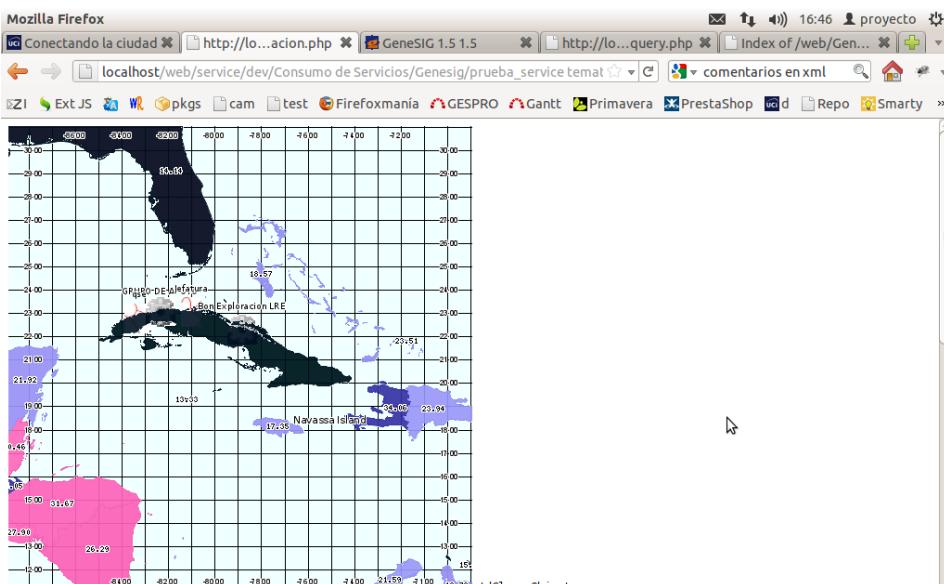


Ilustración 71 Tematización de tipo Corocromático y Coropleta

Un elemento que se debe tener presente, es que los mapas cuantitativos son muy utilizados para representar todo tipo de temas sociales, económicos y ambientales a nivel nacional, regional y local. Las diferentes entidades requieren representar información cuantitativa de manera georreferenciada, especializada y exacta. Por tanto pueden ser considerados como el resultado de la aplicación de ciencias cuánticas y experimentales, con el apoyo principalmente en la estadística descriptiva. Estos mapas generalmente representan datos puntuales, lineales, de distribución de área o la combinación de estos.

5.11.9 Perfiles de visibilidad y altura

GeneSIG provee mecanismos para obtener a partir de un MDE un perfil de altura, mediante el cual se pueden realizar una amplia gama de análisis apoyando en gran medida a la toma de decisiones. Actualmente no se soporta la obtención del modelo de visibilidad, sin embargo este puede ser calculado a partir del propio perfil de altura. Teniendo en cuenta su importancia pues los modelos de visibilidad establecen el área que se puede ver desde un punto y, por tanto, el área desde la que puede verse ese punto. Todo esto puede ser útil para el diseño de redes de control, por ejemplo: de incendios forestales. También como criterio a la hora de ubicar infraestructuras desagradables como vertederos, etc. El análisis de cuencas visuales puede utilizarse para la evaluación del impacto visual de actuaciones con efectos negativos sobre el paisaje.

Es posible construir un modelo de visibilidad, donde cada punto tiene asignado un valor proporcional a la extensión de su cuenca visual. Un modelo de este tipo puede servir de base objetiva para la toma de decisiones ya que permite conocer y comparar con fiabilidad la incidencia visual de las alternativas existentes.

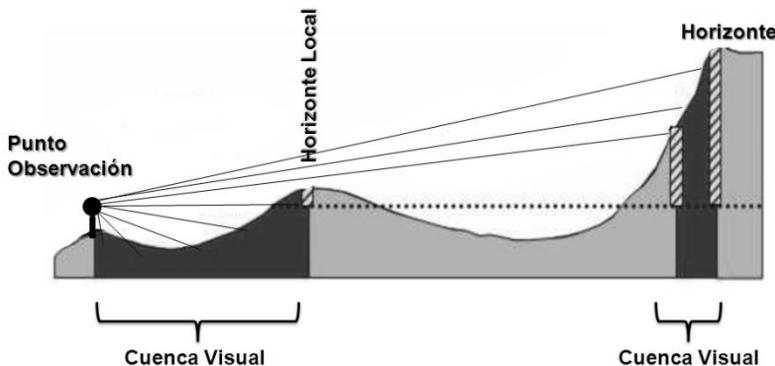


Ilustración 72 Esquema teórico de un perfil de visibilidad y altura

Las estructuras de visibilidad para varios puntos de vista pueden ser definidas por la combinación de la vista y cobertizos de estos puntos de acuerdo a algunos operadores. Combinación de los operadores más comunes son:

- Superposiciones.
- Operadores booleanos.
- Operadores para contar.

La superposición de cuencas visuales contiene más información que cualquier otra estructura. El conjunto de puntos de vista generalmente se limita a un subconjunto de los vértices del MDE. Cualquier estructura de visibilidad puede ser codificada ya sea en forma continua o discreta. Una representación permanente de las particiones de la cuenca visual de cada celda del MDE en sus partes visibles e invisibles.

A continuación se muestra un ejemplo que ilustra como es posible obtener el perfil de altura, una vez especificado un listado de puntos referenciados geográficamente.

```
<?php
$map = new MapRequest( 'ejem.geoweb' );

$map->perfilRequest = new stdClass();
$map->perfilRequest->className = 'PerfilSoapRequest'
$map->perfilRequest->Factor = 900;
$map->perfilRequest->Coordenadas[] = '-76.79145254280515,20.037291347546496,0';
$map->perfilRequest->Coordenadas[] = '-76.20456309939901,20.083871493010506,61589.73624507361';
$map->perfilRequest->Coordenadas[] = '-76.28539944794059,20.106741842557025,353918';

$mapResult = $soapCliente->getMap($map);
echo '<pre>';
print_r( $mapResult->perfilResult );
?>
```

Observese que en la propiedad *Coordenadas* se almacena un listado de estructuras en forma de cadena, las cuales se componen por tres atributos separados por comas con las siguiente organización [*longitud*, *latitud*, *distancia acumulada*]. Otro elemento importante es el *Factor*, este atributo define cada cuantos metros se obtendrá un valor real de altura, por tanto identifica el grado de exactitud con respecto al MDE. A continuación se muestra la salida generada por este plugin.

```
[SerialArea] => Array
(
    [0] => Array
```

```

(
    [0] => 0
    [1] => 1288
    [2] => -76.79145254280515
    [3] => 20.037291347546496
)

[1] => Array
(
    [0] => 900
    [1] => 1316
    [2] => -76.782821815696
    [3] => 20.037976349686
)
...
[394] => Array
(
    [0] => 353918
    [1] => 740
    [2] => -76.28539944794059
    [3] => 20.106741842557025
)
)

[SerialPunto] => Array
(
    [0] => Array
    (
        [0] => 0
        [1] => 1288
        [2] => -76.79145254280515
        [3] => 20.037291347546496
    )

    [1] => Array
    (
        [0] => 61589.73624507361
        [1] => 536
        [2] => -76.20456309939901
        [3] => 20.083871493010506
    )

    [2] => Array
    (
        [0] => 353918
        [1] => 740
        [2] => -76.28539944794059
        [3] => 20.106741842557025
    )
)

```

Como bien se puede apreciar la salida está compuesta por dos listados que hacen referencia a las series numéricas que puedes obtenerse a partir de la selección de puntos definida por el cliente. La serie de puntos corresponde con los previamente seleccionados, a diferencia de la de área, la cual contiene más información en función de obtener un grafico mas preciso. Ambas series están conformadas por una estructura que describe la información que contienen en cada elemento, las cuales se estructuran en el siguiente orden [*distancia acumulada, valor de altura, longitud, latitud*]

5.12 Integración con plataformas externas

Bien es conocido que cada día aparecen nuevas tendencias que aumentan la complejidad de los entornos de las tecnologías de la información. Los productos de software hoy en

día se encuentran en el núcleo de las empresas y de esta forma seguir el mismo modelo de actividad que éstas se imponen, es decir, deben ser reactivos frente a los cambios. Se persigue poner en relación los diferentes sistemas para que funcionen de una forma heterogénea, permitiendo conseguir una solución global que responda a las necesidades del usuario. Gracias a la integración entre sistemas, se optimizan sus procesos de negocio y se disminuyen costos por concepto de tiempo de desarrollo. Le permite integrar nuevas tecnologías a su entorno de soluciones de software, permitiendo alcanzar de una manera más eficiente los objetivos.

5.12.1 Openlayer

En este acápite se expondrá los elementos necesarios para desarrollar un sistema que permita integrar los servicios básicos provistos por la plataforma GeneSIG. En función de garantizar una rápida implementación de un visor de mapas se emplea la librería *Openlayer*. Este ejemplo consta de dos ficheros y el directorio en que se encontrara alojada la mencionada biblioteca, el *script* del servidor que funciona como controlador frontal no se incluye porque puede estar publicado en dependencia del entorno de despliegue, para mayor comprensión véase la figura que se muestra a continuación.

	OpenLayers	18 elementos	Enlace hacia carpeta	jue 12 may 2011 09:36:25 CDT
	index.html	340 bytes	documento HTML	mar 04 sep 2012 14:01:13 CDT
	main.js	1,6 KiB	programa en JavaScript	mar 04 sep 2012 14:29:13 CDT

Ilustración 73 Taxonomía del ejemplo #1 de integración

Contenido del fichero *index.html*, el cual se responsabiliza de definir los elementos bases que conforman la página. Básicamente se define un contenedor en el cuerpo de la misma y se procede a incluir los *script* necesarios para construir la interfaz de la aplicación. En este caso se indica la *url* donde se encuentra el *OpenLayer*, librería empleada para desarrollar la lógica relacionada con el visor de mapas.

```
<html>
  <head>
    <title> Ejemplo #1 de integración entre GeneSIG y OpenLayer </title>
    <link href="OpenLayers/theme/default/style.css" type="text/css" rel="stylesheet" />

    <script src="OpenLayers/OpenLayers.js"></script>
    <script src="main.js"></script>
  </head>

  <body onload="main()">
    <div id="mapContainer"/>
  </body>
</html>
```

A continuación se muestra el contenido del fichero *main.js*, el cual se responsabiliza de la construcción del mapa. Nótese en el fragmento que todo el código se encuentra encapsulado dentro de la función denominada *main*. Esto se debe a que en el cuerpo de la página se le definió la invocación de dicho método una vez culminada su carga, a través del evento *onload*. Básicamente se define la capa base que contendrá la información geográfica, así como el objeto Map y sus propiedades.

```
var main = function()
{
  var layBase = new OpenLayers.Layer.MapServer(
    "GeneSIG services",
    "http://localhost/web/service/FrontGIS.php",
  {
```

```

        layers: []
    },
    isBaseLayer: 1,
    gutter: 0,
    buffer: 0,
    singleTile: true,
    transitionEffect: 'resize'
}
);

var map = new OpenLayers.Map( "mapContainer", {
    allOverlays: true,
    maxResolution: 'auto',
    numZoomLevels: 9,
    controls: [
        new OpenLayers.Control.Navigation(),
        new OpenLayers.Control.KeyboardDefaults(),
        new OpenLayers.Control.LayerSwitcher(),
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.mousePosition()
    ],
    projection: 'init=epsg:4267',
    maxExtent: new OpenLayers.Bounds( -88, 15.7331838565, -65, 28.2668161434 ),
});
map.addLayer( layBase );
map.zoomToMaxExtent();
}

```

En este ejemplo se incorporan algunos de los controles definidos por el OpenLayer en función de facilitar la interacción con el mapa, tal es el caso del controlador de capas, navegación, visor de coordenadas geográficas asociadas al cursor del mouse, etc. Para mayor comprensión véase la figura que se muestra a continuación.



Ilustración 74 Ejemplo #1 de integración

Contenido del fichero *FrontGIS.php*, el cual tiene como responsabilidad generar un mapa a partir de una serie de datos que recibe en la propia petición, tales como el tamaño de imagen, la exención territorial, etc. En otros términos funciona como controlador frontal, encargado de centralizar las peticiones. Entre los elementos parametrizados se encuentra la *url* de publicación para la plataforma GeneSIG, teniendo en cuenta que esto puede variar en función del entorno de despliegue y de igual forma se debe especificar el *mapid*. Otros atributos requeridos son las proporciones de la imagen como resultado de la generación del mapa, así como el *bbox*. Estos últimos juegan un papel fundamental en la

interacción con OpenLayer principalmente si se utiliza el mecanismo de visualización de mapas basado en *tile*, en el cual se subdivide la imagen principal en pequeñas cuadrículas de menor extensión y proporción.

```
<?php
//... build request params ...
$url = isset($_REQUEST['url']) ? $_REQUEST['url'] : 'localhost/web/GeneSIG';
$mapid = isset($_REQUEST['mapid']) ? $_REQUEST['mapid'] : 'ejem.geoweb';
$_REQUEST['imgxy'] = !isset($_REQUEST['imgxy']) ? "640 480" : $_REQUEST['imgxy'];
$_REQUEST['mapext'] = !isset($_REQUEST['mapext']) ? "-88.870862 15.386740 -68.962302 26.758788"
                           : $_REQUEST['mapext'];
$extent = explode(" ", $_REQUEST['mapext']);
$imgsize = explode(" ", $_REQUEST['imgxy']);

try {
    //... build soap cliente object ...
    $wsdl = "http://$url/htdocs/cartoserver.wsdl.php?mapId=$mapid&".time();
    $soapCliente = new SoapClient($wsdl);
    //... define map request object ...
    $map = new stdClass();
    $map->mapId = $mapid;
    //... define location request object ...
    $map->locationRequest->className = 'LocationRequest';
    $map->locationRequest->locationType = 'bboxLocationRequest';
    $map->locationRequest->bboxLocationRequest->className = "BboxLocationRequest";
    $map->locationRequest->bboxLocationRequest->bbox->minx = $extent[0];
    $map->locationRequest->bboxLocationRequest->bbox->miny = $extent[1];
    $map->locationRequest->bboxLocationRequest->bbox->maxx = $extent[2];
    $map->locationRequest->bboxLocationRequest->bbox->maxy = $extent[3];
    $map->locationRequest->bboxLocationRequest->showRefMarks = true;
    //... define images request object ...
    $map->imagesRequest->className = "ImagesRequest";
    $map->imagesRequest->scalebarStatus = 1;
    //... define mainmap object ...
    $map->imagesRequest->mainmap->className = "Image";
    $map->imagesRequest->mainmap->isDrawn = true;
    $map->imagesRequest->mainmap->width = $imgsize[0];
    $map->imagesRequest->mainmap->height = $imgsize[1];
    $map->imagesRequest->mainmap->angle = 0;
    $map->imagesRequest->mainmap->path = "";
    $map->imagesRequest->mainmap->outputFormat = "";
    //... define keymap object ...
    $map->imagesRequest->keymap = new stdClass();
    $map->imagesRequest->keymap->className = "Image";
    $map->imagesRequest->keymap->isDrawn = true;
    $map->imagesRequest->keymap->width = 100;
    $map->imagesRequest->keymap->height = 100;
    $map->imagesRequest->keymap->angle = 0;
    $map->imagesRequest->keymap->path = "";
    $map->imagesRequest->keymap->outputFormat = "";
    //... define scalebar object ...
    $map->imagesRequest->scalebar = new stdClass();
    $map->imagesRequest->scalebar->className = "Image";
    $map->imagesRequest->scalebar->isDrawn = true;
    $map->imagesRequest->scalebar->width = 100;
    $map->imagesRequest->scalebar->height = 100;
    $map->imagesRequest->scalebar->angle = 0;
    $map->imagesRequest->scalebar->path = "";
    $map->imagesRequest->scalebar->outputFormat = "";
    //... execute service ...
    $soapCliente->__setCookie('PHPSESSID', 'frontgis');
    $result = $soapCliente->getMap($map);
    //... show out ...
/*
echo "<img src='http://$url/htdocs/{$result->imagesResult->keymap->path}' > ";
echo "<img src='http://$url/htdocs/{$result->imagesResult->scalebar->path}' > <br>";
echo "<img src='http://$url/htdocs/{$result->imagesResult->mainmap->path}' > <br>";
*/
header("location: http://$url/htdocs/{$result->imagesResult->mainmap->path}");
}

```

```

} catch (SoapFault $error) {
    echo "Error: {$error->faultcode} >> {$error->faultstring} <br>";
}
?>

```

En este caso no se empleo la sintaxis de declaración estándar para clases definidas para el lenguaje *php*, permitiendo reducir algunas líneas de código y minimizar la complejidad de entendimiento. Sin embargo el equipo de desarrollo aconseja que en el consumo de servicios se proceda a desarrollar un paquete que contenga las declaraciones de cada uno de los objetos de tipo *Request*, en aras de ganar por concepto de reutilización y claridad en el código fuente, pues en escenarios más complejos el efectos podría ser todo lo contrario a este ejemplo.

5.12.2 Openlayer y ExtJs

En el acápite anterior se describe los pasos elementales para la construcción de un sistema sencillo que pueda representar un mapa a partir de la política de consumo de servicios. Sin embargo no es común que tengamos que enfrentarnos al desarrollo de sistemas con tan poca complejidad, pues para el mismo cumpla su objetivo en función de apollar a la toma de desiciones, se hace necesario vincular los datos cartográficos con socio-económicos, y de esta forma llevar a cabo análisis mas completos.

Teniendo en cuenta estos elementos a continuación se detallan los pasos necesarios para construir un productos que sin dejar de ser simple, permita gestionar además información socio-económica. Para ello se propone emplear ExtJs, permitiendo confeccionar interfaces más complejas y evidentemente al incrementarse el número de ficheros en este ejemplo se hace necesario definir un esquema organizativo un poco más robusto, para mayor comprensión véase la figura que se muestra a continuación.

libs	2 elementos carpeta	mar 04 sep 2012 16:17:45 CDT
ExtJs-3.2.0	15 elementos Enlace hacia carpeta	mar 31 ene 2012 16:14:24 CST
OpenLayers	18 elementos Enlace hacia carpeta	jue 12 may 2011 09:36:25 CDT
scripts	3 elementos carpeta	mar 04 sep 2012 17:12:13 CDT
main.js	114 bytes programa en JavaScript	mar 04 sep 2012 17:10:51 CDT
msMap.js	919 bytes programa en JavaScript	mar 04 sep 2012 17:09:10 CDT
views.js	1,6 KiB programa en JavaScript	mar 04 sep 2012 17:03:40 CDT
index.html	757 bytes documento HTML	mar 04 sep 2012 17:12:56 CDT

Ilustración 75 Taxonomía del ejemplo #2: Integración basada en Openlayer y ExtJs

A continuación se muestra el contenido del fichero *index.html*, que a diferencia del ejemplo #1 de integración descrito en el acápite anterior, se elimina el componente *html* del cuerpo de la pagina, delegándose a la clase de gestión de interfaces graficas de usuario declarada en el fichero *view.js*. También se procede a incluir los script necesario para utilizar la librería ExtJs.

```

<html>
    <head>
        <title> Ejemplo #2 de integración entre GeneSIG, OpenLayer y ExtJs </title>
        <!-- lib OpenLayers -->
        <link rel="stylesheet" href="libs/OpenLayers/theme/default/style.css" type="text/css" />

        <script src="libs/OpenLayers/OpenLayers.js"></script>
        <!-- lib ExtJs -->
        <link rel="stylesheet" type="text/css" href="libs/ExtJs-3.2.0/resources/css/ext-all.css" />
        <script type="text/javascript" src="libs/ExtJs-3.2.0/adapter/ext/ext-base.js"></script>

```

```

<script type="text/javascript" src="libs/ExtJs-3.2.0/ext-all.js"></script>
<!-- local script -->
<script src="scripts/views.js"></script>
<script src="scripts/msMap.js"></script>
<script src="scripts/main.js"></script>
</head>

<body onload="main()">
</body>
</html>

```

Como bien se puede apreciar en el fragmento de código que se muestra a continuación, el fichero *main.js* reduce considerablemente la cantidad de líneas de código, limitándose únicamente a la instanciación de las clases *AppGUI* y *MsMap*. En este caso se separan las responsabilidades de la construcción de la interfaz gráfica de usuario y el mapa.

```

var main = function()
{
    var gui = new AppGUI ("mapContainer");
    var map = new MsMap("mapContainer", "http://localhost/web/service/FrontGIS.php");
    map.show();
}

```

Comparando con el ejemplo mostrado en el acápite anterior resulta que el fichero *views.js* es un artefacto que se genera nuevo, el cual contiene la implementación de la clase *AppGUI*. Esta es la responsable de la construcción de las interfaces gráficas de usuario. Un elemento importante a resaltar es que se le debe especificar por parámetro el identificador que tomara el contenedor del mapa, en función de que se pueda incrustar dicho componente.

```

var AppGUI = function(mapid)
{
    var _this = this;
    this.westtree = new Ext.tree.TreePanel({
        useArrows: true,
        autoScroll: true,
        animate: true,
        enableDD: true,
        containerScroll: true,
        border: false,
        root: {
            nodeType: 'async',
            text: 'Ext JS',
            draggable: false,
            id: 'src'
        }
    });

    this.eastpanel = new Ext.TabPanel({
        border: false,
        activeTab: 1,
        tabPosition: 'bottom',
        items: [
            {
                html: '<p>A TabPanel component can be a region.</p>',
                title: 'A Tab',
                autoScroll: true
            },
            new Ext.grid.PropertyGrid({
                title: 'Property Grid',
                closable: true,
                source: {
                    "(name)": "Properties Grid",
                    "grouping": false,
                    "autoFitColumns": true,
                    "productionQuality": false,

```

```

        "created": new Date(Date.parse('10/15/2006')),
        "tested": false,
        "version": 0.01,
        "borderWidth": 1
    }
}),
};

this.viewport = new Ext.Viewport(
{
    layout: 'border',
    frame: true,
    items: [
        {
            region: 'south',
            height: 70,
            collapsed : true,
            collapsible: true,
            title: 'South',
            html: 'region sur'
        },
        {
            region: 'east',
            split: true,
            minSize: 300,
            maxSize: 400,
            width: 300,
            collapsible: true,
            title: 'East panel',
            margins: '0 0 0 0',
            layout: 'fit',
            items: [_this.eastpanel]
        },
        {
            region: 'west',
            split: true,
            minSize: 100,
            maxSize: 200,
            collapsible: true,
            title: 'West panel',
            margins: '0 0 0 0',
            items: [_this.westree]
        },
        {
            region: 'center',
            html: '<div id="'+mapid+'"/>'
        }
    ]
});
}

```

Como bien se puede observar en la imagen que se muestra a continuación, la información se estructura a través de un componente base denominado *viewport*, el cual organiza la información por regiones, definiéndose la parte central del mismo en función de alojar el componente visor de mapa y en los laterales denominados east/west para mostrar información socio-económica relacionada con los datos cartográficos.

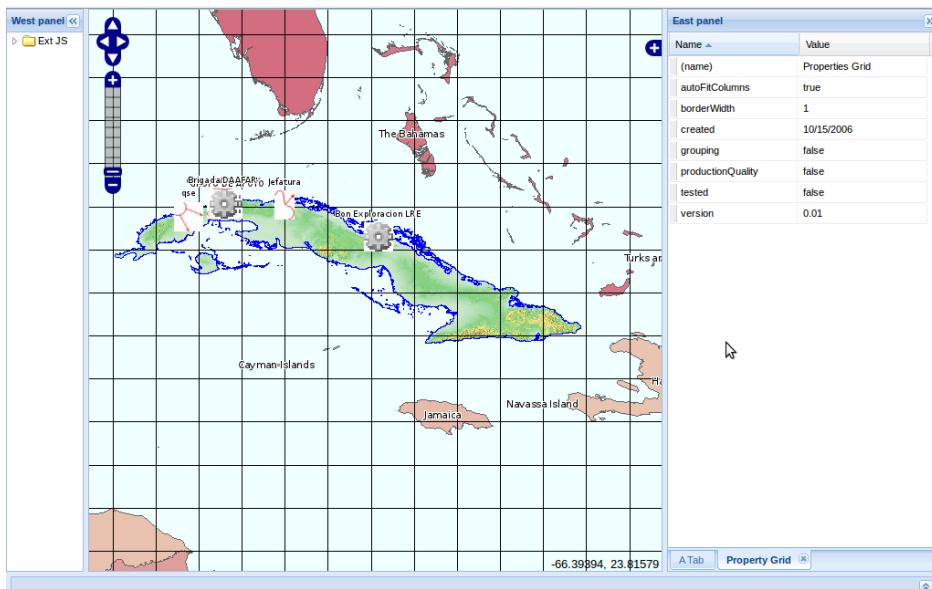


Ilustración 76 Vista del ejemplo #2: Integración basada en Openlayer y ExtJs

A continuación se muestra el contenido del fichero *msMap.js*, el cual contiene la implementación de la clase *MsMap*, responsable de la construcción del mapa.

```

var MsMap = function(mapid, urlserver)
{
    var _this = this;
    this.map = new OpenLayers.Map(mapid,{
        allOverlays: true,
        maxResolution: 'auto',
        numZoomLevels: 9,
        controls: [
            new OpenLayers.Control.Navigation(),
            new OpenLayers.Control.KeyboardDefaults(),
            new OpenLayers.Control.LayerSwitcher(),
            new OpenLayers.Control.PanZoomBar(),
            new OpenLayers.Control.mousePosition()
        ],
        projection: 'init=epsg:4267',
        maxExtent: new OpenLayers.Bounds(-88, 15.7331838565022,-65, 28.2668161434978),
    });

    this.layBase = new OpenLayers.Layer.MapServer(
        "GeneSIG services",
        urlserver,
        {
            layers: []
        },
        {
            isBaseLayer:1,
            gutter:0,
            buffer:0,
            singleTile:true,
            transitionEffect:'resize'
        }
    );

    this.show = function()
    {
        _this.map.addLayer(_this.layBase);
        _this.map.zoomToMaxExtent();
    }
}

```

Nótese que este código es muy similar al que se encontraba dentro de la función *main* del

ejemplo descrito en el acápite anterior, la diferencia radica que en este caso se define una clase que permite encapsular dicha implementación, recibiendo por parámetro el identificador del contenedor del componente mapa, siendo más genérico su empleo, potenciado por las ventajas que trae consigo la herencia.

5.12.3 Zeolides



Definido como un conjunto de componentes de software, librerías, herramientas y tecnologías libres integradas que permiten el desarrollo ágil y basado en componentes de aplicaciones web empresariales de múltiples propósitos,

gran complejidad, así como grandes volúmenes de datos, centrándose el desarrollo en el negocio, los requerimientos y las interfaces de usuario.

Zeolides usa un estilo arquitectónico híbrido, combina el estilo arquitectónico N-Capas y el MVC, tomando lo mejor de los dos, divide la funcionalidad de una aplicación en tres componentes fundamentales: modelo, vista y controlador. A su vez estructura la arquitectura de la aplicación en diferentes capas: presentación, negocio, acceso a datos y datos, para mayor comprensión obsérvese la figura que se muestra a continuación.



Ilustración 77 Estilo arquitectónico de trabajo Zeolides

Otro elemento de su estilo arquitectónico es la incorporación de requerimientos no funcionales a las aplicaciones sin necesidad de modificar el código de la aplicación utilizando los principios de la programación orientada a aspectos (AOP), estos requerimientos tienen diferentes denominaciones según algunos autores: elementos arquitectónicamente significativos, componentes verticales, marco de la arquitectura, entre otros, pero en este documento les llamaremos simplemente aspectos

arquitectónicos.

Por último y no menos importante el uso de inversión de control (IoC) para separar las responsabilidades de una aplicación en componentes, integrando estos a través de contratos bien definidos sin necesidad de instanciar directamente sus clases o conocer sus implementaciones, esto nos permite sentar las bases para el desarrollo de software basado en componentes.

En función de aprovechar todas las bondades y lograr un mayor nivel de integración con los sistemas desarrollados sobre la tecnología antes mencionada, se decide crear un componente para dicho marco de trabajo que facilite la comunicación con GeneSIG. La cual puede ser utilizada tanto desde la capa de presentación, como de la lógica de negocio realizando invocaciones mediante el ICO o utilizando el servicio que devuelve el objeto *SoapClient* para realizar peticiones directamente, el resto del procedimiento es exactamente el mismo al explicado en acápitres anteriores.

5.13 Proyección para la versión 2.0

6. Openlayer

OpenLayers es una biblioteca de JavaScript de código abierto, distribuida una derivación de la licencia BSD para mostrar mapas interactivos en los navegadores web. Ofrece un API para acceder a diferentes fuentes de información cartográfica en la red: *Web Map Services*, Mapas comerciales (tipo Google Maps, Bing, Yahoo), *Web Features Services*, distintos formatos vectoriales, mapas de OpenStreetMap, etc. Inicialmente fue desarrollado por MetaCarta en Junio del 2006. Desde el noviembre del 2007 este proyecto forma parte de los proyectos de *Open Source Geospatial Foundation*. Actualmente el desarrollo y el soporte corre a cargo de la comunidad de colaboradores.

6.1 Elementos basicos

Los mapas de OpenLayers se muestran a través de sistemas web, por tanto lo primero que se debe que realizar en función de crear un proyecto básico basado en dicha biblioteca, consiste en crear una página en lenguaje *HTML* donde albergaremos el resto de elementos. Un esquema de página web sencilla podría ser el siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hola Mundo</title>
  </head>

  <body>
  </body>
</html>
```

Lo siguiente es añadir un elemento *div* dentro del cual se mostrará el mapa. El elemento elegido deberá tener definido su atributo *id* y además deberá tener definidas al menos las propiedades de estilo *width* y *height*. Hay que señalar que estos atributos se deben especificar dentro de una cláusula de estilo CSS, pues si asignamos los valores correspondientes a la anchura y la altura del elemento contenedor a través del *DOM*, se

corre el riesgo de que no se visualice el contenido definido para el mapa.

Por tanto si se le incorporan al fragmento anterior los elementos expuestos, especificandosele un contenedor de mapa denominado *divMap*, entonces quedaría de la siguiente forma:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hola Mundo</title>
    <style type="text/css">
      #divMap {
        width: 100%;
        height: 515px;
        border: solid 2px #808080;
      }
    </style>
  </head>

  <body>
    <div id="divMap"></div>
  </body>
</html>
```

Ahora hay que añadir una pequeña rutina *Javascript* que utilice las clases de OpenLayers que permitan mostrar el mapa. Para poder utilizar dichos recursos debemos cargar la librería en memoria. Esto se hace añadiendo un *link* en la cabecera de la página al fichero que implementa la librería y que generalmente se denomina igual. Este lo podemos tener nosotros en nuestro dominio, lo que nos garantizará que siempre utilizamos la misma versión, o podemos enlazar directamente con la versión última disponible en el sitio de OpenLayers. Para lograr esto último el enlace que se debe añadir dentro de la declaración del *head* sería de la siguiente forma:

```
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
```

De lo contrario en la propiedad denominada *src* se le debe especificar la ruta relativa con respecto al fichero que gestiona la demanda. Una vez cargada la librería en memoria podemos acceder todos los recursos brindados por la biblioteca.

Para mostrar un mapa se utiliza la clase *OpenLayers.Map*, a la que se le deben añadir una o más capas de información cartográfica. El objeto resultante del proceso de instanciación es el que mantiene la proyección en la que estamos trabajando, el centro del mapa y el nivel de zoom en cada momento. Además se responsabiliza por la gestión de los controles asociados al mismo, siendo este uno de los recursos que garantizan altos niveles flexibilidad e interacción.

```
var mapa = new OpenLayers.Map("divMap");
```

Como bien se había mencionado en acápitres anteriores la parte visible del mapa, lo constituye el conjunto de capas han sido añadidas al mismo. Por tanto OpenLayers dispone de una clase genérica denominada *OpenLayers.Layer*, de la cual derivan los tipos específicos de capas adaptados a las distintas fuentes de datos. En caso que se requiera consumir una fuente dedatos proveida a partir de un servicio de mapas se debería especificar una instancia de esta clase de la siguiente forma:

```
var laywms = new OpenLayers.Layer.WMS( "DPA",
  "http://www.idee.cu/wms/dpa",
```

```
    { layers: 'all', projection: 'EPSG:4230' }  
);
```

Nótese que el primer parámetro especificado al constructor es un nombre que la identifica y que será visualizado en el control definido para la gestión de la visualización de capas. Por otra parte el segundo define la dirección *URL* del servicio y el último es un objeto que describe las opciones de la petición *GETMAP*, de acuerdo con las prescripciones del estándar OGC para servicios de mapas. Por ultimo sólo queda añadir la capa y fijar la posición del centro del mapa, así como el nivel de zoom inicial, lo que hacemos mediante el método *OpenLayers.Map.zoomToMaxExtent()*. A continuación se muestra el ejemplo que contempla los elementos anteriormente mencionados.

```
<html>  
  <head>  
    <script src="http://www.openlayers.org/api/OpenLayers.js"></script>  
    <script type="text/javascript">  
      var init = function () {  
        var map = new OpenLayers.Map('mapa');  
        var laywms = new OpenLayers.Layer.WMS( "DPA",  
          "http://www.idee.cu/wms/dpa",  
          {layers: 'all', projection: 'EPSG:4230'}  
        );  
        mapa.addLayer(laywms);  
        mapa.zoomToMaxExtent();  
      }  
    </script>  
  </head>  
  <body onload="init()">  
    <div id="mapa" style="width: 600px; height: 300px"></div>  
  </body>  
</html>
```

A partir del fragmento de código anterior se genera una salida similar a la que ilustra la imagen que se muestra a continuación.



Ilustración 78 Ejemplo de capa que ilustra la DPA de Cuba

Para aprender más acerca del desarrollo sobre OpenLayers, lo mejor sería descargarse la librería y sus códigos fuente desde el sitio oficial. Pese a que la documentación en línea de no es del todo completa, dentro del paquete se encuentra la descripción y el código fuente de las clases con sus atributos y métodos perfectamente documentados. Además se incluye una colección muy completa de ejemplos que sirven como tutorial en función de utilizar las distintas técnicas definidas por esta biblioteca.

6.1.1 Tipos básicos de datos

OpenLayers está escrita en el lenguaje *Javascript*, por tanto dispone de los mismos tipos de datos. Para el cual según la especificación vigente hasta la fecha existen cinco tipos de datos primitivos: *Undefined*, *Null*, *Boolean*, *Number* y *String*. Además se dispone de una colección de objetos nativos: *Object*, *Boolean*, *Error*, *SyntaxError*, *Function*, *Number*, *EvalError*, *TypeError*, *Array*, *Date*, *RangeError*, *URIError*, *String*, *RegExp*, y *ReferenceError*. Teniendo en cuenta los elementos expuestos esta biblioteca ofrece un conjunto de clases con métodos útiles para operar los distintos tipos de datos básicos:

OpenLayers.String: Esta clase proporciona los siguientes métodos en función de la manipulación de cadenas de textos:

- **startsWith (string, substr):** Comprueba si la cadena *string* comienza con la cadena *substr*. Devuelve un resultado de tipo *boolean*.
- **contains(str, substr):** Comprueba si la cadena *substr* está contenida dentro de la cadena *str*. Devuelve valor *true* en caso afirmativo y *false* en caso negativo.
- **trim(str):** Devuelve una cadena en la que se han eliminado todos los espacios que pudiera haber al principio o al final de la cadena *str*.
- **camelize(str):** Convierte una cadena a base de guines en el convenio Camelize (Camel-Case). Por ejemplo la cadena '*lat-punto*' la convierte en *latPunto* y la cadena *-lat-punto* en '*LatPunto*'. Devuelve una cadena con las modificaciones, sin alterar el contenedor original.
- **isNumeric(str):** Devuelve un valor de tipo *boolean* si la caadena corresponde a un número. Reconoce el formato exponencial para números reales. Por ejemplo *isNumeric("2.5e3")* devuelve *true*.
- **numericIf(str):** Si la cadena *str* es un valor numérico devuelve un objeto de tipo *Number* con ese valor. En otro caso devuelve un *String* con la misma cadena recibida.

OpenLayers.Number: Esta clase posee dos propiedades que se utilizan para dar formato a los números: '*decimalSeparator*' y '*thousandsSeparator*'. Además contiene las siguientes funciones para manipular datos tipo numericos:

- **limSigDigs(float, sig):** Redondea el valor del '*float*' al número de decimales indicado por el *integer* denominado *sig*. Devuelve un valor de tipo *Float* con el número redondeado.
- **format(num, dec, tsep, decsep):** Devuelve un *string* con el *float* denominado *num* redondeado al número de decimales indicados por el *integer* nombrado *dec*. Como separadores de millares y decimales se utilizarán los String '*tsep*' y '*decsep*' respectivamente.

Para el trabajo con funciones y arreglos define las clases denominadas *OpenLayers.Function* y *OpenLayers.Array*, a las cuales les incorpora a través del prototipo una serie de utilidades que abstraen a su manipulación. Por otra parte define la clase *OpenLayers.Date* para la gestión del recuso fecha, así como *OpenLayers.Class* como soporte del API en sentido general y *OpenLayers.Element* paa describir los elementos del DOM.

En función al trabajo con mapas se definen

6.1.2 La clase OpenLayers.Util

La clase OpenLayers.Util es una colección de métodos y propiedades utilitarios que permiten realizar determinadas tareas frecuentes de un modo sencillo. La documentación de la clase la puedes encontrar en: <http://dev.openlayers.org/docs/files/OpenLayers/Util-js.html>.

Desarrollamos a continuación la explicación de algunos de los métodos de la clase Util:

```
GetElement()  
OpenLayers.Util.getElement(  
    idElemento : String [, idElemento2 : String, ... ] )  
    : [ domElement o Array[ domElements ] ]
```

Admite como parámetro uno o más id's correspondientes a elementos del DOM que queremos seleccionar. Nos devuelve un elemento o un Array con los elementos solicitados.

El caso habitual con un solo parámetro de entrada es equivalente a `document.getElementById(idElemento)` o al clásico `$(idElemento)`. Un ejemplo de utilización sería:

```
var element = OpenLayers.Util.getElement("panellzquierdo");
```

En este caso la variable '`element`' recibira una referencia al elemento del documento html cuyo id es '`panellzquierdo`'.

isElement

```
OpenLayers.Util.isElement( elemento : Object ) : Boolean
```

Recibe como parámetro el elemento a analizar y devuelve `TRUE` si se trata de una instancia de la clase Element y `FALSE` en caso contrario.

extend

```
OpenLayers.Util.extend( dest: Object, source: Object ) : Object
```

Este método copia las propiedades del objeto '`source`' en el objeto destino '`dest`', sin modificar propiedades que no se especifiquen en el objeto '`source`'. Es muy útil en situaciones en las que creamos un objeto con las propiedades por defecto, y luego con el método '`extend()`' afinamos el valor de las propiedades que consideremos adecuado.

Un ejemplo de utilización lo tenemos en la creación de un estilo de visualización para una Feature punto. Podemos crear un estilo con los valores por defecto de la siguiente manera:

```
// Crear un objeto estilo con valores por defecto  
var point_style = OpenLayers.Util.extend({},  
    OpenLayers.Feature.Vector.style['default']);
```

Ahora solo necesitamos modificar los valores de las propiedades que queramos personalizar:

```
point_style.strokeColor = "#000011";  
point_style.fillColor = "#ffa500";  
  
removeItem()
```

```
OpenLayers.Util.removeItem( miArray: Array, item: Object ) : Array
```

Esta función recorre los elementos del Array 'miArray' y elimina el elemento 'item' si existe. La función devuelve una referencia al Array modificado. Para vaciar un Array hay que utilizar 'miArray.length = 0 '.

6.1.3 La clase OpenLayers.Map

OpenLayers.Map es la clase fundamental de la librería OpenLayers. Todo programa de OpenLayers tiene como objeto crear un mapa que se visualizará en pantalla. Los objetos de la clase OpenLayers.Map son una colección de capas, OpenLayers.Layer, que contienen la información que se quiere mostrar, y controles, OpenLayers.Control, que permiten interactuar con el usuario. El objeto Map también es el responsable de gestionar la visualización del mapa en cuanto a Zoom y Panning se refiere.

El constructor de la clase Map tiene la siguiente forma:

```
OpenLayers.Map( div : [DOMElement/ String], options: Object)
```

El primer parámetro es una referencia al elemento 'div' del documento html destinado a contener el mapa. En lugar de una referencia se puede pasar una cadena de texto con el 'id' del elemento. El segundo parámetro es un Array de opciones en la forma 'key:value'. Las opciones son valores de las propiedades del objeto 'Map' que queramos fijar con un valor determinado en el constructor. Un ejemplo de mapa sin opciones adicionales podría ser:

```
var map = new OpenLayers.Map("divMapa");
```

Podemos añadir las opciones adicionales directamente en el constructor:

```
var map = new OpenLayers.Map("divMapa", {  
    projection: 'EPSG:4326',  
    units: 'm'  
});
```

Pero también podemos preparar primero el objeto de opciones y añadirlo luego al constructor por referencia:

```
var opciones = { projection: 'EPSG:4326', units: 'm' };  
var map = new OpenLayers.Map("divMapa", opciones);
```

La clase OpenLayers.Map expone una buena colección de propiedades y métodos. Básicamente un objeto Map es una colección de capas (Layer) y controles (Control). Veamos en primer lugar una serie de propiedades que son colecciones de objetos pertenecientes al mapa:

- **layers:** La propiedad 'layers' es un Array de objetos OpenLayers.Layer, que contiene la colección de capas del mapa. Para gestionar la colección de capas se dispone de los métodos: addLayer(), addLayers(), removeLayer(), getNumLayers(), getLayerIndex(), setLayerIndex(), raiseLayer(), setBaseLayer(), getLayer(), getLayersBy(), getLayersByClass(), getLayersByName(), setLayerZIndex(), resetLayersZIndex().

- **controls:** Colección de controles asociados al mapa. Para manejar la colección se utilizan los métodos: addControl(), addControls(), addControlToMap(), getControl(), getControlsBy(), getControlsByClass() y removeControl().
- Métodos del objeto Map

6.1.4 Jerarquias de clases con OpenLayers

6.1.5 Forms que ejecutan Javascript

6.2 Controles

Los controles se utilizan para interactuar con el mapa. Permiten hacer zoom, mover el mapa, conocer las coordenadas del cursor, dibujar features, etc. En OpenLayers V2.10 hay 40 clases de controles para utilizar con los mapas. De ellos, dos está desaconsejado su uso pues se van a suprimir en la próxima versión de OpenLayers, la versión 3.0. Los controles descatalogados son '*MouseDefaults*' y '*MouseToolBar*'. De los 38 controles restantes, dos son nuevos de esta versión, '*SLDSelect*' y '*WMTSGetFeatureInfo*'. El resto vienen de versiones anteriores, si bien el control '*OpenLayers.Control.Panel*' ha sufrido modificaciones en su funcionamiento respecto de anteriores versiones.

Todos los controles derivan de la clase '*OpenLayers.Control*', y todos los controles se añaden al objeto '*OpenLayers.Map*', directa o indirectamente. La clase '*OpenLayers.Map*' tiene una propiedad 'controls' que guarda la lista de controles del mapa. Para añadir un control al mapa se puede hacer mediante el método '*OpenLayers.Map.addControl()*' o bien directamente en el constructor de '*OpenLayers.Map*'.

El constructor de la clase OpenLayers.Map permite añadir controles en el momento de la creación del mapa. Si no se indica nada, el mapa añade los siguientes controles por defecto:

- *OpenLayers.Control.Navigation*
- *OpenLayers.Control.PanZoom*
- *OpenLayers.Control.ArgParser*
- *OpenLayers.Control.Attribution*

Esto es, si creamos el mapa con el siguiente constructor:

```
var map = new OpenLayers.Map("divMapa");
```

Automáticamente quedan añadidos al mapa los controles por defecto indicados. Podemos crear un mapa sin ningún control de la siguiente manera:

```
var map = new OpenLayers.Map("divMapa", {controls: []});
```

Si lo que queremos es añadir una serie de controles elegidos por nosotros habría que invocar el constructor del mapa de la siguiente manera:

```
var map = new OpenLayers.Map("divMapa", {
  controls: [
    new OpenLayers.Control.PanZoom(),
    new OpenLayers.Control.Attribution()
  ]
});
```

También podemos crear un mapa sin controles y añadírselos después:

```
var map = new OpenLayers.Map("divMapa", { controls: []});
var ctrlPanZoom = new OpenLayers.Control.PanZoom();
map.addControl(ctrlPanZoom);
var ctrlAttribution = new OpenLayers.Control.Attribution();
map.addControl(ctrlAttribution);
```

En ambos casos hemos creado un mapa con los controles '*PanZoom*' y '*Attribution*'. Debemos cuidar siempre de añadir el control '*Attribution*' para mostrar correctamente los *Attribution* de los mapas que utilicemos.

- En OpenLayers las definiciones de estilo por defecto de controles y otros elementos se declaran y definen en la hoja de estilo
- <http://www.openlayers.org/api/theme/default/style.css>. Podemos añadir un enlace a dicha hoja de estilo en nuestro código:
- <link type="text/css" rel="stylesheet" href="http://www.openlayers.org/api/theme/default/style.css"/>

Si echamos un vistazo al fichero vemos que se definen propiedades de estilo para todo tipo de elementos que son fáciles de identificar por su nombre de clase. También podemos personalizar la posición o el aspecto de los controles como se verá en capítulos posteriores

La lista de controles que podemos utilizar es la siguiente:

- Controles generales
 - ArgParser
 - Attribution
 - Button
 - Graticule
 - KeyboardDefaults
 - LayerSwitcher
 - OverviewMap
 - Panel
 - PanPanel
 - PermaLink
 - SLDSelect (nuevo versión 2.10)
 - Snapping
 - Split
- Zoom, Panning, Position
 - DragPan
 - MousePosition
 - Navigation
 - NavigationHistory
 - NavToolBar
 - Pan
 - PanZoom
 - PanZoomBar
 - ZoomBox
 - ZoomIn

- ZoomPanel
- ZoomOut
- ZoomToMaxExtent
- Features
 - EditingToolbar
 - DragFeature
 - DrawFeature
 - GetFeature
 - ModifyFeature
 - SelectFeature
 - TransformFeature
- Medir
 - Measure
 - Scale
 - ScaleLine
- Servicios de Mapas
 - WMSGetFeatureInfo
 - WMTSGetFeatureInfo (nuevo versión 2.10)
 - En próximos capítulos iremos explicando la utilización detallada de cada uno de estos controles.

6.2.1 La clase OpenLayers.Control

Propiedades:

- id : (String)
- map : (OpenLayers.Map) El mapa al que pertenece el control
- div : (DOMElement) Elemento 'div' que alberga el control
- type : (Number) Especifica el comportamiento del control, dentro del panel. Puede tomar los valores:
 - OpenLayers.Control.TYPE_BUTTON :

6.2.2 Utilización de los controles

OpenLayers.Control.KeyboardDefaults

Este control permite hacer zoom mediante las teclas '+' y '-' y mover el mapa con las teclas de flecha. Para activar el control solo hay que crear una instancia de la clase *OpenLayers.Control.KeyboardsDefaults* y añadirla a la colección de controles del mapa.

```
var ctrl = new OpenLayers.Control.KeyboardsDefaults();
map.addControl(ctrl);
```

OpenLayers.Control.Attribution

En OpenLayers la clase *OpenLayers.Layer* tiene una propiedad llamada 'attribution' que es un String que se muestra en pantalla cuando el mapa dispone del control *OpenLayers.Control.Attribution*. El constructor de '*OpenLayers.Map*' añade un control 'Attribution' por defecto si no se especifica lista de controles en el constructor. Si al crear el mapa añadimos nuestra propia lista de controles debemos cuidar el detalle de añadir el

control OpenLayers.Control.Attribution El control 'Attribution' dispone entre otras de las siguientes propiedades y métodos que pueden ser de utilidad:

- `draw()` : Inicializa y dibuja el 'attribution'. Devuelve una referencia al elemento 'div' que contiene el control. El valor de retorno podríamos utilizarlo para ajustar propiedades de estilo, posición, etc.
- `updateAttribution()` : Actualiza el 'attribution' en pantalla. (El valor de la cadena attribution esta en `OpenLayers.Layer.attribution`).

En OpenLayers las definiciones de estilo por defecto de controles y otros elementos se declaran y definen en la hoja de estilo <http://www.openlayers.org/api/theme/default/style.css>. Podemos añadir un enlace a dicha hoja de estilo en nuestro código:

```
<link type="text/css" rel="stylesheet"> href="http://www.openlayers.org/api/theme/default/style.css" </link>
```

Si echamos un vistazo al fichero vemos que se definen propiedades de estilo para todo tipo de elementos que son faciles de identificar por su nombre de clase. El nombre de la clase que controla el aspecto del control 'Attribution' es '`olControlAttribution`'. Por defecto se define de la siguiente manera:

```
.olControlAttribution {  
    font-size: smaller;  
    right: 3px;  
    bottom: 4.5em;  
    position: absolute;  
    display: block;  
}
```

Podemos redefinir los valores a conveniencia, siempre respetando la regla de oro: Colocar el attribution en un lugar visible del mapa.

OpenLayers.Control.Scale

El control Scale muestra la escala actual del mapa, en forma de ratio (1: 10000). El constructor tiene la siguiente firma:

```
OpenLayers.Control.Scale ( element: DOMElement, options: Object)
```

El aspecto y posición del control Scale viene definido por su atributo 'class' que es '`olControlScale`'. Los valores de estilo por defecto para este control los podemos encontrar en el fichero style.css y son los siguientes:

```
.olControlScale {  
    right: 3px;  
    bottom: 3em;  
    display: block;  
    position: absolute;  
    font-size: smaller;  
}
```

Vemos que el control se situa en la esquina inferior derecha por defecto. Estas posiciones las podemos modificar a nuestra conveniencia personalizando las cláusulas de estilo de la clase '`olControlScale`'. El control Scale tiene una propiedad 'geodesic', que es un Boolean que indica si las distancias se deben calcular mediante geodésicas o cartesianas. Por defecto este valor es '`FALSE`', valor que se recomienda para la utilización con mapas en

'EPSG:4326' (WGS84). En el caso de trabajar con mapas en Spherical Mercator, 'EPSG:900913', se recomienda establecer el valor de 'geodesic' en 'TRUE'. En este caso, la escala se calcula en base al tamaño horizontal del pixel en el centro de la ventana de visualización del mapa. El control Scale proporciona dos métodos:

- **draw()**: Dibuja el control. Devuelve el DOMElement donde se aloja el control.
- **updateScale()**: Recalcula y muestra la escala del mapa.

Para añadir el control Scale a un mapa debemos crear una instancia del control y añadirla a la colección de controles del mapa.

```
var ctrl = new OpenLayers.Control.Scale();
map.addControl(ctrl);
```

OpenLayers.Control.Button

Este control representa un botón de pulsar. Está pensado para formar parte de un control contenedor como 'Panel'. La clase Button deriva de Control, a la que añade un valor prefijado para la propiedad 'type' y un método llamado 'trigger'. El tipo de control está fijado en el valor predefinido OpenLayers.Control.Type_BUTTON.

El único método que aporta la clase Button es el método *trigger()*. El método 'trigger' será llamado por el panel contenedor cada vez que se pulse el botón. Para crear un botón hay que utilizar el constructor heredado de la clase Control, que admite como parámetro un objeto 'options' que será un Array asociativo con los valores de las propiedades que queramos especificar. En el caso de los botones le deberemos pasar como parámetros el nombre de clase CSS del 'div' donde se alojará el botón, así como una referencia a la función 'trigger()' que hay que llamar cuando se pulse el botón. Un ejemplo podría ser el siguiente :

```
var button = new OpenLayers.Control.Button({
    displayClass: "MyButton", trigger: myFunction
});
panel.addControls([button]);
```

El programa utilizará como nombre de la clase CSS del botón la cadena pasada como argumento añadiéndole el sufijo 'ItemInactive'. Esto es, en nuestro ejemplo la clase CSS del botón será 'MyButtonItemInactive'. En este mismo ejemplo, el programa llamará a la función disparadora 'myFunction' cada vez que se pulse el botón.

6.2.3 Botones personalizados

La clase OpenLayer.Control.Panel sirve para agrupar varios controles tipo botón o pulsador. Si no creamos un elemento 'div' en la página html para alojar el Panel, el programa le asignará uno y lo situará en la esquina superior izquierda del mapa. Para poder controlar la posición y el aspecto de nuestro Panel hay que crear un elemento 'div' en la página html. En el elemento 'div' hay que definir un 'id' y una clase 'class':

```
<div class="miPanel" id="panel0"></div>
<div id="mapa"></div>
```

Podemos situar el Panel dentro o fuera del mapa. La posición y aspecto del Panel la controlamos mediante cláusulas de estilo CSS. En nuestro ejemplo hemos decidido situar el Panel en la parte superior y fuera del mapa. La definición de estilo para el Panel:

```
.miPanel {
    position: relative;
    left: 103px;
    height: 38px;
    width: 78px;
    margin: 6px;
    border: solid 2px blue;
    background-color: #00d0ff;
}
```

Para completar el proceso de creación del Panel habrá que llamar al constructor de la clase en el código Javascript. El constructor de la clase Panel admite un único parámetro que es un Object con las propiedades que queramos especificar al control. En nuestro caso debemos indicarle cual es el elemento 'div' asignado al Panel. Lo hacemos en dos pasos. Primero recuperamos el elemento 'div' con el método 'OpenLayers.Util.getElement' y luego se lo pasamos al constructor del Panel:

```
// Cogemos una referencia al 'div' del panel
var el = OpenLayers.Util.getElement('panel0');

// Creamos el panel0 pasandole la referencia del 'div'
var panel = new OpenLayers.Control.Panel({ 'div': el });
```

Con esto queda completado el proceso de crear nuestro Panel para alojar otros controles. A los paneles se les pueden añadir cualquiera de los controles predefinidos para hacer zoom, pan, dibujar features, etc. En este caso vamos a añadir dos botones de la clase genérica OpenLayers.Control.Button.

La clase OpenLayers.Control.Button ofrece funcionalidad para un botón alojado en un Panel. Para crear un botón no es necesario crear ningún elemento 'div' dentro del código html de la página. El programa se encargará de hacerlo por nosotros. Nosotros lo que tenemos que hacer es indicarle el nombre de la clase 'class' que define nuestros botones. A la cadena que le pasemos al constructor, por ejemplo 'btn1', el programa le añade el sufijo 'ItemActive', de modo que el nombre de la clase del botón queda 'btn1ItemActive'. Este será el nombre de clase que habrá que utilizar en la hoja de estilo para definir el aspecto de nuestro botón. En nuestro ejemplo hemos definido dos botones: 'btn1' y 'btn2'. Los nombres de clase serán 'btn1ItemActive' y 'btn2ItemActive'.

La clase Button tiene una propiedad llamada 'trigger' (disparador) que apunta a una función que será llamada cada vez que se pulse el botón. Este es el segundo parámetro que es necesario pasarle al constructor y deberá ser el nombre de la función que queramos que se ejecute cuando pulsemos el botón. Con estas consideraciones los constructores de nuestros dos botones quedarán:

```
btn1 = new OpenLayers.Control.Button({
    displayClass:"btn1",
    trigger: btn1Click
});
btn2 = new OpenLayers.Control.Button({
    displayClass:"btn2",
    trigger: btn2Click
});
```

El primer parámetro es 'displayClass', que como hemos indicado es el prefijo del nombre de clase del botón. El segundo parámetro es el método que se llamará al pulsar el botón.

```
// Función 'trigger' del botón 1
function btn1Click() {
    var el = document.getElementById('mapa');
    el.innerHTML = "Botón 1";
```

```

        }
    // Función 'trigger' del botón 2
    function btn2Click() {
        var el = document.getElementById('mapa');
        el.innerHTML = "Botón 2";
    }
}

```

En cuanto a la definición del estilo de los botones hemos elegido la siguiente:

```

.btn1ItemInactive {
    float: left; width: 32px; height:32px; margin: 2px;
    border: solid 1px #333333;
    border-bottom: solid 2px #333333;
    border-right: solid 2px #333333;
    background-color: #0069b4;
    background-image: url('image1.gif');
    background-repeat: no-repeat;
    background-position: center
    cursor: pointer;
}
.btn2ItemInactive {
    float: left; width: 32px; height:32px; margin: 2px;
    border: solid 1px #333333;
    border-bottom: solid 2px #333333;
    border-right: solid 2px #333333;
    background-color: #0069b4;
    background-image: url('movie.gif');
    background-repeat: no-repeat;
    background-position: center
    cursor: pointer;
}

```

Observese que se han utilizado los nombres de clase mencionados anteriormente para cada botón, usando la pseudo clase 'hover' para variar el color del fondo cuando el ratón sobrevuela el botón. Se ha definido para cada botón 'float: left' para que se vayan adosando de izquierda a derecha en lugar de apilarse de arriba a abajo. Además se ha especificado el tamaño y un borde más grueso abajo y a la derecha para dar sensación de volumen. Se ha situado una imagen de fondo del botón, centrada y sin repetición. Creados los botones solo falta añadirlos al Panel y añadir el Panel al Map:

```

// Añadimos los botones al panel
panel.addControls([btn1, btn2]);

// Añadimos el panel al mapa
map.addControl(panel);

```

6.3 Capas

Las capas se definen como contenedores de información. En el caso de OpenLayer son más bien los gestores de dicha información. Se encargan de refenciar los datos cartográficos, traerlos al cliente y renderizarlos.

6.3.1 La clase OpenLayers.Layer

La clase OpenLayers.Layer es la clase base para todos los tipos de capas especializadas, que son las que realmente se añaden a los mapas. El constructor de la clase OpenLayers.Layer tiene la siguiente signatura:

```
OpenLayers.Layer( name: String, options: Object)
```

El primer parámetro es una cadena que permitirá identificar a la capa, por ejemplo en el control LayerSwitcher. El segundo parámetro es un Hashtag de opciones adicionales que se añadirán a la capa. La forma de este parámetro es un Array asociativo de parejas 'key:value' separadas por comas. El constructor no se utiliza directamente, pues se utilizan las clases derivadas, pero un ejemplo ilustrativo de la forma de pasar el segundo parámetro podría ser el siguiente:

```
var capa = new OpenLayers.Layer(  
    "L1",  
    {opacity: 0.5, isBaseLayer: false }  
)
```

La clase OpenLayers.Layer expone las siguientes propiedades:

- **id** : String .- El valor del atributo 'id' asignado al elemento 'div' de la capa
- **name** : String .- El nombre de la capa es una cadena que permite identificar a la capa en algunas situaciones, por ejemplo en el Control LayerSwitcher
- **div** : DOMElement .- Es una referencia al elemento 'div' que alberga la capa.
- **opacity** : Float .- Es un número entre 0 (= transparente) y 1 (= opaco) que indica el grado de transparencia de la capa.
- **alwaysInRange** : Boolean .- Se debe establecer en 'true' cuando la visualización de la capa no se debe basar en zoom.
- **events** : OpenLayers.Events .- La propiedad 'events' es la colección de eventos de la capa. Es una referencia a un objeto de la clase OpenLayers.Events.
- **map** : Es una referencia al mapa que contiene a la capa. Se establece en la función addLayer() del mapa o en la función setMap() de la capa. El objeto apuntado es un OpenLayers.Map.
- **isBaseLayer** : Es un Boolean que indica si se trata de una capa base. Por defecto es falso. Las clases derivadas especializadas establecen un valor por defecto para cada tipo de capa.
- **alpha** : Es un Boolean que indica si las imágenes de la capa tienen canal alfa. Por defecto es false.
- **displayInLayerSwitcher** : Boolean que indica si el nombre de la capa debe de aparecer o no en el control LayerSwitcher. El valor por defecto es 'true'.
- **visibility** : Es un Boolean que indica si la capa es visible o no. El valor por defecto es 'true'.
- **attribution** : Se trata de la cadena, String, que se mostrará en el control Attribution.
- **inRange** : Es un Boolean que indica si el valor de la resolución actual está entre el mínimo y el máximo de la capa (minResolution, maxResolution). Se establece cada vez que cambia el zoom.
- **imageOffset** : Desplazamientos x,y debidos al borde, 'gutter', en las imágenes

que tienen 'gutter'. Es un objeto OpenLayers.Pixel.

- **options** : Se trata de un objeto mediante el cual se pueden pasar al constructor de la capa valores iniciales para cualquiera de las propiedades de la capa. Vease un ejemplo mas arriba en la explicación del constructor de Layer.
- **eventListeners** : (Object)
- **gutter** : (Integer) El valor del ancho del borde, si lo tiene. Por defecto es cero
- **projection** : (Object)
- **Objeto OpenLayers.Projection** con la proyección de la capa. Si se pasa en el objeto 'options' del constructor, se puede pasar como una cadena del tipo 'EPSG:4326', pero durante la creación de la capa se construirá un objeto Projection. Cuando se utiliza esta opción suele ser necesario fijar también 'maxExtent', 'maxResolution' y 'units'.
- **units** : (String). Las unidades de medida de la capa. Por defecto son grados, y la variable 'units' tiene el valor 'degrees'. Los valores posibles son: 'degrees' (o 'dd'), 'm', 'ft', 'km', 'mi', 'inches'
- **scales** : (Array). Un array con las escalas del mapa para cada zoom en orden descendente. Este parámetro sólo tiene sentido si está bien calibrado con la resolución concreta del monitor en el que estemos trabajando. Además debe de estar bien definida la propiedad 'units' de la capa. En general es preferible utilizar la propiedad 'resolutions'.
- **resolutions** : (Array). Un array con las resoluciones del mapa para cada zoom en orden descendente. La resolución es el numero de unidades de mapa por pixel. Si no se especifica al construir la capa, se calcula en base a otras propiedades de la capa (maxExtent, maxResolution, maxScale, etc).
- **maxExtent** : (OpenLayers.Bounds). The center of these bounds will not stray outside of the viewport extent during panning. In addition, if displayOutsideMaxExtent is set to false, data will not be requested that falls completely outside of these bounds.

6.3.2 Tipos de capas

OpenLayers.Layer.Markers OpenLayers nos ofrece una capa especial para alojar nuestros marcadores: OpenLayers.Layer.Markers. Un marcador es una instancia de la clase OpenLayers.Marker, que es una combinación de un ícono y una posición. El constructor de la clase OpenLayers.Markers tiene la siguiente forma:

```
OpenLayers.Markers ( name : String, options: Object)
```

El primer parámetro es una cadena que permitirá identificar a la capa, por ejemplo en el control LayerSwitcher. El segundo parámetro es una Hashtable de opciones extra que queramos asignar a la capa. Las propiedades públicas de la capa Markers son:

- **isBaseLayer** : (Boolean = false) Las capas de marcadores no son capas base. Se sobreescribe la propiedad de la clase base OpenLayers.Layer con el valor false.

- **markers** : (Array(OpenLayers.Marker)) Lista de marcadores de la capa. Esta propiedad es un Array que guarda las referencias a los marcadores que se han ido añadiendo a la capa.
- **drawn** (Boolean) : Nos indica si se ha dibujado la capa. En algunas situaciones al inicializar la capa o hacer zoom es necesario comprobar esta variable y si es necesario llamar al método 'draw()'
- **La capa Markers proporciona los siguientes métodos:**
- **addMarker(marker : OpenLayers.Marker)** : Añade a la capa el marcador referenciado por la variable 'marker'
- **removeMarker(marker : OpenLayers.Marker)** : Elimina de la capa el marcador 'marker'.
- **drawMarker(marker : OpenLayers.Marker)** : Calcula la posición pixel del marcador, lo crea y lo añade al 'div' de la capa.
- **clearMarkers()**: Esta función deja vacía la lista de marcadores de la capa. Los marcadores en sí mismos no se eliminan, pero si se retiran de la lista de marcadores de la capa.
- **setOpacity(opacity: float)**: Establece la opacidad de todos los marcadores de la capa. (0 = transparente; 1= opaco)
- **getDataExtent()** : Devuelve un OpenLayers.Bounds con la extensión que abarca todos los marcadores de la capa
- **moveTo (bounds, zoomChanged, dragging)** :
- **Puedes** consultar la documentación completa de la clase OpenLayers.Layer.Markers en el siguiente enlace: <http://dev.openlayers.org/docs/files/OpenLayers/Layer/Markers-js.html>
OpenLayers.Layer.Vector Las capas vectoriales están pensadas para alojar 'features' vectoriales, que serán instancias de las clases derivadas de OpenLayers.Feature.Vector. El constructor de la clase OpenLayers.Layer.Vector admite dos parámetros:

OpenLayers.Layer.Vector (name : String, options: Object); El parámetro 'nombre' es una cadena de texto que sirva para identificar la capa y el parámetro 'options', que es opcional, es un objeto con propiedades de la capa que queramos establecer en un valor distinto del que se asigna por defecto. Una vez creada la capa se le pueden ir añadiendo las features que queramos visualizar mediante el método 'addFeatures()' que admite como parámetro un Array de features. Dichas 'features' habrán sido creadas previamente y serán instancias de la clase OpenLayers.Feature.Vector. La geometría y el estilo de visualización asignados a la feature serán las que marquen la forma en que se visualizarán las features en la capa. La capa, una vez creada, se añade al mapa mediante el método OpenLayers.Map.addLayer(). Un ejemplo de una capa vectorial con un punto podría ser:

```
var ptgeom = new OpenLayers.Geometry.Point(-3.7856, 42.2540);
var ptstyle = OpenLayers.Util.extend({}, 
    OpenLayers.Feature.Vector.style['default']);
var ptfeat = new OpenLayers.Feature.Vector(
    ptgeom, null, ptstyle);
var layerVector = new OpenLayers.Layer.Vector("Capa Vectorial");
layerVector.addFeatures([feat]);
map.addLayer(layerVector);
```

- OpenLayers.Layer.Boxes

La capa WMS

Las capas OpenLayers.Layer.WMS están pensadas para albergar los datos provenientes de consultas a los Web Map Services de acuerdo con las especificaciones del OGC (Open Geospatial Consortium). En las siguientes referencias puedes obtener una introducción a los Web Map Services y otros temas relacionados:

- TutorialWMS.pdf : Introducción a los WMS.
- GetCapabilities.pdf : Capacidades del servicio.
- GetMap.pdf : La petición GetMap.
- GetFeatureInfo.pdf : Petición de fenómenos.
- Inspire.pdf : Directiva INSPIRE.
- Esta capa hereda de OpenLayers.Layer.Grid. Para acceder a un WMS lo primero que debemos conocer es la url del servicio de mapas. Por ejemplo en España disponemos de un WMS del Instituto Geográfico en la siguiente dirección url:

<http://www.idee.es/wms/IDEE-Base/IDEE-Base>

Existen numerosos servicios de mapas para consultar en la red. En este enlace se pueden ver algunos de ellos. Hay que especificar una serie de parámetros que definirán los datos que queremos recibir del servicio:

Eventos del objeto Map

El objeto OpenLayers.Map es capaz de gestionar los siguientes eventos:

- *preaddlayer* Se dispara antes de añadir una capa. El objeto 'event' tiene una propiedad 'layer' que referencia la capa que se va a añadir.
- *addlayer triggered* Despues de crearse la capa. El objeto 'event' tiene una propiedad 'layer' que referencia la capa que se va a añadir.
- *removelayer* Se dispara despues de la creación de una capa. El objeto 'event' tiene una propiedad 'layer' que referencia la capa que se va a añadir.
- *changelayer* Despues de cambiar el nombre, el orden, la opacidad, los parámetros o la visibilidad de una capa. Los manejadores del evento recibirán un objeto 'event' con unas propiedades 'layer' y 'property' con referencias a la capa cambiada y la clave de la propiedad cambiada (name, order, opacity, params, visibility).
- *movestart* Este evento se dispara al comienzo de un drag, pan o zoom.
- *move* Despues de cada drag, pan o zoom.
- *moveend* Se dispara al completar un drag, pan o zoom.
- *zoomend* Se dispara cuando se completa un zoom.
- *addmarker* Despues de añadir un marcador.
- *removemarker* Se dispara despues de eliminar un marcador.
- *clearmarkers* Despues de borrar los marcadores.
- *mouseover* Despues de sobrevolar el ratón el mapa.
- *mouseout* Disparado al salir el cursor del mapa.
- *mousemove* Cuando el cursor se mueve sobre el mapa.
- *dragstart* No funciona. Utilizar movestart.
- *drag* No funciona. Utilizar move.

- *dragend* No funciona. Utilizar *moveend*.
- *changebaselayer* Se dispara cuando se cambia la capa base.
- Para utilizar alguno de los eventos debemos definir una función que admita un parámetro de tipo OpenLayers.Event, y que será la encargada de gestionar el evento.
- function miManejador(evt) {
.....
}

• El objeto 'evt' que recibe la función al dispararse el evento tendrá distintas propiedades según el evento de que se trate. Todos tienen al menos las propiedades 'object', una referencia al objeto que dispara el evento ('map.events.object') y 'element', una referencia al elemento DOM que dispara el evento ('map.events.element'). Los eventos del navegador tienen además la propiedad 'xy' con un objeto OpenLayers.Pixel con las coordenadas del punto donde se ha disparado el evento, relativas a la ventana del mapa.
Para registrar nuestro manejador en la lista de manejadores del objeto Map debemos utilizar el método *register*' de la clase OpenLayers.Events.
- map.events.register(type, obj, listener);
- Los parámetros que le pasamos son :
 - *type* Un String con el nombre del evento que queremos manejar.
 - *obj* El objeto que recibe el evento. Normalmente el objeto Map
 - *listener* Una referencia a la función manejadora del evento
 - Hemos desarrollado un ejemplo en el que añadimos un manejador para el evento 'mousemove' y mostramos las coordenadas de cursor en pixels y en grados en unos elementos 'text' fuera del mapa.
testMapEvents.html
Si desplazas el mapa o haces zoom podrás comprobar que las coordenadas pixel siempre se refieren a la esquina de la ventana del mapa, mientras que las coordenadas longitud-latitud se refieren a la posición absoluta en el mapa.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Test Map Events</title>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script type="text/javascript">
var map, layer;
function init() {
  map= new OpenLayers.Map("map");
  var wms = new OpenLayers.Layer.WMS( "OpenLayers WMS",
  "http://labs.metacarta.com/wms/vmap0?", {
  layers: 'basic'});
  map.addLayer(wms);
  map.setCenter(new OpenLayers.LonLat(-5.5651,36.0),4);
  map.events.register("mousemove",map,mouseMoveHandler);
}
}
```

```

function mouseMoveHandler(evt) {
var el = document.getElementById("txt");
var pos=this.events.getMousePosition(evt);
el.value=pos;
el = document.getElementById("txt2");
el.value=map.getLonLatFromPixel(pos);
}
</script>
</head>
<body onload="init()">
<h5 style="color: green">Manejando los eventos del objeto OpenLayers.Map</h5>
<div id="map" style="width: 640px; height: 400px; border: solid 1px #a06600"></div>
<input type="text" size="30" value="no set" id="txt"/>
<input type="text" size="30" value="no set" id="txt2"/>

</body>
</html>

```

Coordenadas del cursor

En OpenLayers disponemos de una gestión de eventos a través del objeto OpenLayers.Map que nos permite interactuar con el usuario de diversas maneras. Uno de los eventos proporcionado por el 'Map' es 'mousemove' que nos proporciona las coordenadas en pixels del cursor cuando este se mueve por el mapa. Para utilizar el evento debemos definir una función 'callback' que será llamada cada vez que se dispare el evento, esto es, cada vez que se mueva el cursor sobre el mapa. Además debemos 'registrar' el evento de forma que indiquemos: el evento que queremos gestionar, el objeto 'Map' y la función que hay que llamar cada vez que se dispare el evento.

`map.events.register("mousemove", map, mouseMoveHandler);`

La función que reciba el evento debe de admitir un parámetro 'event'. Por ejemplo podría ser algo parecido a lo siguiente:

```

function mouseMoveHandler(e) {
    var position = this.events.getMousePosition(e);
    var lonlat = map.getLonLatFromPixel(position);
}

```

Vemos que el evento nos proporciona las coordenadas pixel, (a través del método 'getMousePosition' del objeto 'event'). Mediante el método 'getLonLatFromPixel' del objeto 'map' podemos obtener las coordenadas del punto en la proyección que esté utilizando el mapa. Esto en algunas ocasiones no nos resuelve el problema. Pongamos por caso que estamos utilizando el mapa OpenStreetMap que trabaja en la proyección Spherical Mercator. En este caso el método 'getLonLatFromPixel' nos dará las coordenadas en la proyección Spherical Mercator. Si lo que queremos son las coordenadas Longitud y Latitud en el sistema WGS84, por ejemplo, habrá que transformar las coordenadas desde Spherical Mercator a WGS84. Para ello podemos utilizar el método 'transform' de la clase LonLat, aunque este método transforma las coordenadas del punto original. Nosotros hemos preferido definir una función de transformación, que mediante un 'clon', devuelve un punto con las coordenadas transformadas sin alterar el punto original:

```

function transformToWGS84( sphMercatorCoords) {
    // Transforma desde SphericalMercator a WGS84
    // Devuelve un OpenLayers.LonLat con el pto transformado
    var clon = sphMercatorCoords.clone();
    var pointWGS84= clon.transform(
        new OpenLayers.Projection("EPSG:900913")
        new OpenLayers.Projection("EPSG:4326"));
    return pointWGS84;
}

```

La coordenadas del punto devuelto son 'float' con los decimales que le correspondan. Podemos necesitar gestionar el formato de los números. Nosotros hemos decidido poner las coordenadas con cuatro decimales, para lo que utilizamos el siguiente método:

```

function transformMouseCoords(lonlat) {
    var newlonlat=transformToWGS84(lonlat);
    var x = Math.round(newlonlat.lon*10000)/10000;
    var y = Math.round(newlonlat.lat*10000)/10000;
    newlonlat = new OpenLayers.LonLat(x,y);
    return newlonlat;
}

```

Los valores los vamos a mostrar en un elemento 'div' llamado 'coords'. Incorporando estos afinamientos a nuestra rutina de gestión del evento, podría quedar algo así:

```

function mouseMoveHandler(e) {
    var position = this.events.getMousePosition(e);
    var lonlat = map.getLonLatFromPixel(position);
    OpenLayers.Util.getElement("coords").innerHTML =
        transformMouseCoords(lonlat);
}

```

Hemos puesto todo junto en una página que puedes consultar en el siguiente enlace:

```

<!DOCTYPE html>
<html>
<head>
<title>TestEvents1</title>
<style type="text/css">
.smallmap {
width: 700px;
height: 450px;
}
#coords {
color: blue;
}
</style>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script type="text/javascript">
var map;
var centerWGS84, centerOSM;
var projWGS84, projSphericalMercator;
var osmLayer;

```

```

function init(){

projWGS84 = new OpenLayers.Projection("EPSG:4326");
projSphericalMercator = new OpenLayers.Projection("EPSG:900913");

centerWGS84=new OpenLayers.LonLat(-3.69357,40.41062);
centerOSM = transformToSphericalMercator(centerWGS84);

map = new OpenLayers.Map("map");

osmLayer = new OpenLayers.Layer.OSM();

map.addLayer(osmLayer);
map.setCenter(centerOSM, 15);
map.events.register("mousemove", map, mouseMoveHandler);

}

function mouseMoveHandler(e) {
var position = this.events.getMousePosition(e);
var lonlat = map.getLonLatFromPixel(position);
OpenLayers.Util.getElement("coords").innerHTML = transformMouseCoords(lonlat);
}

function transformMouseCoords(lonlat) {
var newlonlat=transformToWGS84(lonlat);
var x = Math.round(newlonlat.lon*10000)/10000;
var y = Math.round(newlonlat.lat*10000)/10000;
newlonlat = new OpenLayers.LonLat(x,y);
return newlonlat;
}

function transformToWGS84( sphMercatorCoords ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = sphMercatorCoords.clone(); // Si no uso un clon me transforma el punto original
var pointWGS84= clon.transform(
new OpenLayers.Projection("EPSG:900913"), // to Spherical Mercator Projection;
new OpenLayers.Projection("EPSG:4326")); // transform from WGS 1984
return pointWGS84;
}

function transformToSphericalMercator( wgs84LonLat ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = wgs84LonLat.clone(); // Si no uso un clon me transforma el punto original
var pointSphMerc= clon.transform(
new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
new OpenLayers.Projection("EPSG:900913")); // to Spherical Mercator Projection;
return pointSphMerc;
}

</script>
</head>
<body onload="init()">
<h1 id="title">MousePosition Control</h1>

```

```

<div id="map" class="smallmap"></div>
<div id="coords"></div>
<p></p>
</body>
</html>

```

Otra forma de mostrar las coordenadas del cursor es a través del control 'MousePosition'. Este control nos muestra las coordenadas en la proyección del mapa. Este control tiene un atributo 'formatOutput' que se puede definir apuntando a un método que transforme las coordenadas que queremos mostrar, como se muestra a continuación:

```

var ctrl = new OpenLayers.Control.mousePosition()
ctrl.formatOutput = transformMouseCoords;
map.addControl(ctrl);

```

Donde como rutina de 'callback' hemos utilizado el método transformMouseCoords definido anteriormente. En la siguiente página utilizamos las dos técnicas simultáneamente y mostramos las coordenadas del cursor en el control y en un elemento HTML:

```

<!DOCTYPE html>
<html>
<head>
<title>TestEvents2</title>
<style type="text/css">
.smallmap {
width: 700px;
height: 450px;
}
#coords {
color: blue;
}
</style>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script type="text/javascript">
var map;
var centerWGS84, centerOSM;
var projWGS84, projSphericalMercator;
var osmLayer;

function init(){

projWGS84 = new OpenLayers.Projection("EPSG:4326");
projSphericalMercator = new OpenLayers.Projection("EPSG:900913");

centerWGS84=new OpenLayers.LonLat(-3.69357,40.41062);
centerOSM = transformToSphericalMercator(centerWGS84);

map = new OpenLayers.Map("map");

var ctrl = new OpenLayers.Control.mousePosition()
ctrl.formatOutput = transformMouseCoords;

```

```

map.addControl(ctrl);

osmLayer = new OpenLayers.Layer.OSM();

map.addLayer(osmLayer);
map.setCenter(centerOSM, 15);
map.events.register("mousemove", map, mouseMoveHandler);

}

function mouseMoveHandler(e) {
var position = this.events.getMousePosition(e);
var lonlat = map.getLonLatFromPixel(position);
OpenLayers.Util.getElement("coords").innerHTML = transformMouseCoords(lonlat);
}

function transformMouseCoords(lonlat) {
var newlonlat=transformToWGS84(lonlat);
var x = Math.round(newlonlat.lon*10000)/10000;
var y = Math.round(newlonlat.lat*10000)/10000;
newlonlat = new OpenLayers.LonLat(x,y);
return newlonlat;
}

function transformToWGS84( sphMercatorCoords ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = sphMercatorCoords.clone(); // Si no uso un clon me transforma el punto original
var pointWGS84= clon.transform(
new OpenLayers.Projection("EPSG:900913"), // to Spherical Mercator Projection;
new OpenLayers.Projection("EPSG:4326")); // transform from WGS 1984
return pointWGS84;
}

function transformToSphericalMercator( wgs84LonLat ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = wgs84LonLat.clone(); // Si no uso un clon me transforma el punto original
var pointSphMerc= clon.transform(
new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
new OpenLayers.Projection("EPSG:900913")); // to Spherical Mercator Projection;
return pointSphMerc;
}

</script>
</head>
<body onload="init()">
<h1 id="title">MousePosition Control</h1>
<div id="map" class="smallmap"></div>
<div id="coords"></div>
<p></p>
</body>
</html>

```

Videomap: Mostrar videos

El objeto OpenLayers.Popup es el utilizado para mostrar un cuadro con información en el mapa, por ejemplo al pulsar sobre un marcador. El constructor tiene la siguiente forma:

```
var pop = new OpenLayers.Popup( id,  
    lonlat,  
    contentSize,  
    contentHTML,  
    closeBox,  
    closeBoxCallback );
```

Los parámetros que se pasan al constructor son:

- id: (String) Identificador único para el Popup
- lonlat: (OpenLayers.LonLat) Coordenadas en el mapa de la esquina superior izquierda del recuadro a mostrar
- contentSize: (OpenLayers.Size) Tamaño del recuadro a mostrar
- contentHTML : (String) Código de marcado HTML que se mostrará en el recuadro
- closeBox: (Boolean) Indica si aparecerá o no un control en X para cerrar el recuadro.
- closeBoxCallback: (Function) La función que se llamará al cerrar el recuadro. (Opcional)
- El parámetro 'contentHTML' puede albergar cualquier cosa que podamos describir con HTML. Son ejemplos habituales mostrar un texto informativo o mostrar un mapa de detalle de la zona pulsada. Nosotros hemos preferido mostrar un video de Youtube. El código de marcado se obtiene en los vídeos de Youtube en la opción 'insert'. Copiando y pegando en el código javascript, con unos pequeños retoques, pues funciona. Hemos cambiado las comillas dobles por comillas simples para ajustar el parámetro. Además hemos ajustado el tamaño al de nuestro recuadro y hemos añadido dos parámetros del API de Youtube: autoplay=1 y rel=0. El parámetro 'autoplay' por defecto vale '0' e indica si se debe reproducir el vídeo automáticamente o no. El parámetro 'rel=0' (por defecto vale '1') indica que no se muestren en la imagen enlaces a otros vídeos parecidos.

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>Video Map</title>  
<style type="text/css">  
.smallmap { width: 700px; height: 450px; border: solid 1px #000000 }  
#coords { color: blue; font-family: Verdana, sans-serif; font-size: 1.0em; font-weight: bold; }  
#title { color: green; font-family: Verdana, sans-serif; font-size: 1.3em; font-weight: bold; }  
</style>  
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>  
<script type="text/javascript">  
var map;  
var centerWGS84, centerOSM; markerPos;  
var projWGS84, projSphericalMercator;  
var osmLayer, markersLayer;  
var marker;  

```

```

function init(){

projWGS84 = new OpenLayers.Projection("EPSG:4326");
projSphericalMercator = new OpenLayers.Projection("EPSG:900913");

centerWGS84=new OpenLayers.LonLat(-3.745,43.465);
centerOSM = transformToSphericalMercator(centerWGS84);

markerPos= transformToSphericalMercator( new OpenLayers.LonLat(-3.785,43.465));

map = new OpenLayers.Map("map");

osmLayer = new OpenLayers.Layer.OSM();

map.addLayer(osmLayer);
map.setCenter(centerOSM, 11);
map.events.register("mousemove", map, mouseMoveHandler);

createMarkers();


}

function createMarkers() {
markersLayer = new OpenLayers.Layer.Markers( "Markers" );
map.addLayer(markersLayer);

var size = new OpenLayers.Size(21,31);
var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);
var icon = new OpenLayers.Icon("http://www.ingemoral.es/img/markers/icons-2/scenic.png",size,offset);
marker=new OpenLayers.Marker(markerPos, icon);
marker.events.register('click', marker, markerClick);
markersLayer.addMarker(marker);

}

function markerSimulClick() {
var el = OpenLayers.Util.getElement("enlace");
marker.events.triggerEvent('click',marker.events);
}

function markerClick(evt) {
var position = this.events.getMousePosition(evt);
var lonlat = evt.object.lonlat;//map.getLonLatFromPixel(position);
if(popup == null){
// Como código de marcado ponemos el proporcionado por youtube
// adaptando el tamaño del video al del popup
// Hemos cambiado las comillas dobles por comillas simples
// y hemos añadido los parametros &autoplay=1&rel=0 para que
// el video se ejecute automáticamente (autoplay=1)
// y no muestre accesos a otros videos similares (rel=0).
// También hemos añadido un título delante.
popup = new OpenLayers.Popup("popup", lonlat,
new OpenLayers.Size(262,160),

```

```

"<span style='color: black; font-weight: bold;'>Santander (Cantabria) </span><br/>"+
"<object width='160' height='96'>"+
"<param name='movie' value='http://www.youtube.com/v/LMOqazDpkvw?fs=1&hl=es_ES'></param>" +
"<param name='allowFullScreen' value='true'></param>" +
"<param name='allowscriptaccess' value='always'></param>" +
"<embed src='http://www.youtube.com/v/LMOqazDpkvw?fs=1&hl=es_ES&autoplay=1&rel=0' " +
" type='application/x-shockwave-flash' allowscriptaccess='always' " +
" allowfullscreen='true' width='246' height='144'></embed></object>",
false);
popup.setBackgroundColor("green");
popup.setBorder("1px solid green");
map.addPopup(popup);
} else{
popup.toggle();
}
OpenLayers.Event.stop(evt);

}

function mouseMoveHandler(e) {
var position = this.events.getMousePosition(e);
var lonlat = map.getLonLatFromPixel(position);
OpenLayers.Util.getElement("coords").innerHTML = transformMouseCoords(lonlat);
}

function transformMouseCoords(lonlat) {
var newlonlat=transformToWGS84(lonlat);
var x = Math.round(newlonlat.lon*10000)/10000;
var y = Math.round(newlonlat.lat*10000)/10000;
newlonlat = new OpenLayers.LonLat(x,y);
return newlonlat;
}

function transformToWGS84( sphMercatorCoords ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = sphMercatorCoords.clone(); // Si no uso un clon me transforma el punto original
var pointWGS84= clon.transform(
new OpenLayers.Projection("EPSG:900913"), // to Spherical Mercator Projection;
new OpenLayers.Projection("EPSG:4326")); // transform from WGS 1984
return pointWGS84;
}

function transformToSphericalMercator( wgs84LonLat ) {
// Transforma desde SphericalMercator a WGS84
// Devuelve un OpenLayers.LonLat con el pto transformado
var clon = wgs84LonLat.clone(); // Si no uso un clon me transforma el punto original
var pointSphMerc= clon.transform(
new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
new OpenLayers.Projection("EPSG:900913")); // to Spherical Mercator Projection;
return pointSphMerc;
}

</script>
</head>

```

```

<body onload="init()">
<h1 id="title">Video Map</h1>
<div id="map" class="smallmap"></div>
<div id="coords"></div>
<a id="enlace" href="javascript:markerSimulClick()">Activar/Desactivar Video</a>
<p></p>
</body>
</html>

```

Map Markers

Google popularizó los marcadores para mapas, esos pequeños iconos que sirven para señalar una posición sobre un mapa electrónico y que suelen proporcionar acceso a alguna característica adicional al pulsar sobre ellos.

Desde entonces han aparecido cientos de marcadores, cada vez más sofisticados, pero que básicamente conservan el 'look' original que permite identificarlos como marcadores.

Actualmente hay muchas colecciones de marcadores disponibles para utilizar en nuestros mapas. Una buena colección es la que se puede ver en Google-Maps-Icons, colección de 900 marcadores de uso libre y organizados por categorías : Cultura, Educación, Transporte, ...

Hay otros sitios de marcadores :

- Mapki : Google Mapki
- Mapito : Free Map Marker Icons
- MapChannels
- También se pueden utilizar imágenes vectoriales para crear marcadores. Una forma de hacerlo es a través de juegos de caracteres personalizados. La calidad de estos marcadores es buena, pero sólo proporcionan dos colores. Independientemente la simbología para cartografía está reglamentada en muchos campos:
- Map Symbols
- Topographic Symbol Chart
- USA Chart Symbols
- Aeronautical Chart Symbols
- http://naco.faa.gov/content/naco/online/pdf_files/8th_IFR_Symbols.pdf Nautical Chart Symbols
- A continuación presentamos el listado de un ejemplo de mapa sencillo con un marcador, desarrollado para la API de Google:
-
- ```

function init()
{
 if(GBrowserIsCompatible())
 {
 var map = new GMap2(document.getElementById("map"));
 map.setCenter(new GLatLng(40.678,-3.97), 13);
 }
}

```

```

 var location=new GLatLng(centerLatitude, centerLongitude);

 map.setCenter(location, startZoom);

 var marker=new GMarker(location);

 map.addOverlay(marker);
 }
}

• El ejemplo siguiente es similar al anterior pero desarrollado para la API de OpenLayers:
• function init()
 {
 var map, layer, center;
 center = new OpenLayers.LonLat(-3.97,40.678);
 map = new OpenLayers.Map("map");
 layer = new OpenLayers.Layer.WMS("Cartociudad",
 "http://www.cartociudad.es/wms/CARTOCIUDAD/CARTOCIUDAD",
 {layers: ['Vial', 'DivisionTerritorial', 'FondoUrbano']});;

 map.addLayer(layer);

 var markers = new OpenLayers.Layer.Markers("Markers");
 map.addLayer(markers);

 var size = new OpenLayers.Size(21,31);
 var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);
 var icon = new OpenLayers.Icon(
 "http://www.ingemoral.es/img/markers/icons-2/scenic.png",
 size,offset);
 markers.addMarker(new OpenLayers.Marker(center, icon));

 map.setCenter(center, 13);
 }
}

```

- El resultado de ambas páginas se puede ver en los siguientes enlaces. También puedes examinar el código fuente completo de los ejemplos.

## Dynamic map markers

Como continuación del artículo sobre Estilos personalizados de puntos mostramos un mapa en el que se utilizan instancias de la clase *OpenLayers.Feature.Vector* con una geometría '*Point*' en la que en lugar de utilizar los gráficos predefinidos utilizamos el atributo '*externalGraphic*' apuntando a cualquier tipo de grafico.

En nuestro ejemplo hemos utilizado algunos gráficos GIF dinámicos a modo de marcadores.

## Elementos vectoriales

Vamos a desarrollar un ejemplo muy simple de mapa al que añadiremos tres elementos (features) vectoriales: un punto, una linea y un polígono. OpenLayers define la clase *OpenLayers.Feature* y sus derivadas como una combinación de geometría y atributos.

En particular la clase *OpenLayers.Feature* es una combinación de un LonLat y un Marker. La clase que utilizaremos en el ejemplo es *OpenLayers.Feature.Vector*. Esta clase proporciona un atributo 'Geometry' que describe su geometría, un atributo 'attributes' para describir el objeto y un atributo 'style' con el que se puede dotar de propiedades de estilo de representación a las features siguiendo las reglas de los Styled Layer Descriptor (SLD) del OGC.

En este artículo utilizaremos el estilo por defecto que OpenLayers asigna, y en un artículo posterior desarrollaremos el tema de los estilos.

El atributo 'geometry' es una referencia a un objeto de la clase *OpenLayers.Geometry*. Esta clase tiene varias clases derivadas que nos permiten definir distintos tipos de geometrías.

En nuestro caso utilizaremos las clases *OpenLayers.Geometry.Point* para definir la geometría de un punto, *OpenLayers.Geometry.LineString* para definir líneas abiertas y *OpenLayers.Geometry.LinearRing* para definir polígonos (líneas cerradas).

En los tres casos el procedimiento seguido para añadir features vectoriales es el mismo. Se crea un objeto con la geometría adecuada que luego es el parámetro que se pasa al constructor de la clase *OpenLayers.Feature.Vector*. En este caso, al utilizar los estilos por defecto, la geometría es el único parámetro que pasamos al constructor de la feature.

Para visualizar las features creadas debemos añadirlas a una capa vectorial del tipo *OpenLayers.Layer.Vector*. A esta capa se le añaden las features a través del método *addFeatures()*.

El programa que desarrolla el ejemplo es el siguiente:

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Capa Vectorial</title>
<script src="../javascript/openlayers/OpenLayers.js" type="text/javascript"></script>
<script type="text/javascript">
function init() {
 var map = new OpenLayers.Map('map');
 var capa = new OpenLayers.Layer.WMS("WMS", "http://labs.metacarta.com/wms/vmap0",
 {layers: 'basic'});
 map.addLayer(capa);
 // Crear la capa
 var pointFeature = new OpenLayers.Feature.Vector(new OpenLayers.Geometry.Point(-4.0, 40.0));
 map.addFeatures([pointFeature]);
}</script>
</head>
```

```

// Crear una feature line
var var pointList = [];
for(var newPoint p=0; p<10; newPoint=newPoint+newPoint)
 newPoint.x = Math.random(1);
 newPoint.y = Math.random(1);
 pointList.push(newPoint);
}
var lineFeature = OpenLayers.Feature.Vector(
 OpenLayers.Geometry.LineString(pointList));

// crear una feature polygon
var for(var var var pointList = [];
 a = p = * p<6; newPoint = OpenLayers.Geometry.Point(
 r = Math.random(1) * Math.cos(a),
 r = Math.random(1) * Math.sin(a));
 pointList.push(newPoint);
 }
 pointList.push(pointList[0]);

var new linearRing = OpenLayers.Geometry.LinearRing(pointList);
var new polygonFeature = OpenLayers.Feature.Vector(
 OpenLayers.Geometry.Polygon([linearRing]));

// Crear una capa vectorial
var vectorLayer = new OpenLayers.Layer.Vector("Capa Vectorial");

// Añadir las features a la capa vectorial
vectorLayer.addFeatures([
 pointFeature,
 lineFeature,
 polygonFeature]);

// Añadir la capa vectorial al mapa
map.addLayer(vectorLayer);

// Visualizar el resultado
var center=new OpenLayers.LonLat(point.x,
 point.y);
map.setCenter(center,4);
}

</script>
</head>
<body>
 <div id="map" style="width:600px; height:300px;"></div>
</body>
</html>

```

El resultado del ejemplo lo puedes ver en el siguiente enlace:

```

<script type="text/javascript">
function init() {
var map;

// Crear la capa base
map = new OpenLayers.Map('map');
var layer = new OpenLayers.Layer.WMS("OpenLayers WMS",
"http://labs.metacarta.com/wms/vmap0", {layers: 'basic'});
map.addLayer(layer);

// Crear una feature punto
var point = new OpenLayers.Geometry.Point(-4.0, 40.0);
var pointFeature = new OpenLayers.Feature.Vector(point);

// Crear una feature line
var pointList = [];
var newPoint = point;
for(var p=0; p<10; ++p) {
newPoint = new OpenLayers.Geometry.Point(newPoint.x + Math.random(1),
newPoint.y + Math.random(1));
pointList.push(newPoint);
}
var lineFeature = new OpenLayers.Feature.Vector(
new OpenLayers.Geometry.LineString(pointList));

// crear una feature polygon
var pointList = [];
for(var p=0; p<6; ++p) {
var a = p * (2 * Math.PI) / 7;
var r = Math.random(1) + 1;
var newPoint = new OpenLayers.Geometry.Point(point.x + (r * Math.cos(a)),
point.y + (r * Math.sin(a)));
pointList.push(newPoint);
}
pointList.push(pointList[0]);

var linearRing = new OpenLayers.Geometry.LinearRing(pointList);
var polygonFeature = new OpenLayers.Feature.Vector(
new OpenLayers.Geometry.Polygon([linearRing]));

// Crear una capa vectorial
var vectorLayer = new OpenLayers.Layer.Vector("Capa Vectorial");

// Añadir las features a la capa vectorial
vectorLayer.addFeatures([pointFeature, lineFeature, polygonFeature]);

// Añadir la capa vectorial al mapa
map.addLayer(vectorLayer);

```

```
// Visualizar
var center=new OpenLayers.LonLat(point.x, point.y);
map.setCenter(center,4);
}
</script>
```

### Estilos personalizados: Puntos

OpenLayers proporciona una clase Vector, derivada de Feature, que encapsula la funcionalidad de una geometría más unas reglas de aspecto visual. La geometría deberá ser una instancia de alguna clase derivada de OpenLayers.Geometry. Las reglas de estilo de representación se asignan en el atributo 'style' de la clase OpenLayers.Feature.Vector. Las reglas de estilo siguen las directrices del W3C para gráficos vectoriales SVG. También son normativas de referencia en este sentido los documentos correspondientes del Open Geospatial Consortium : '*Symbology Encoding Implementation Specification*' y '*Styled Layer Descriptor profile of the Web Map Service Implementation Specification*'. Además la clase Vector hereda de Feature un atributo llamado 'attributes' que permite añadir características adicionales al objeto de que se trate. En este artículo trataremos sobre la forma de dar estilo a las Features cuya geometría es un punto :OpenLayers.Geometry.Point. El constructor de la clase Vector tiene la siguiente forma:

```
Openlayers.Feature.Vector(geometry, attributes, style)
```

El primer atributo que hay que pasar al constructor es una geometría. Definir una geometría de la clase Point es sencillo:

```
var point = new OpenLayers.Geometry.Point(-4.04, 40.68);
```

En nuestro ejemplo no vamos a utilizar el parámetro 'attributes'. El parámetro 'style' deberá ser un objeto de la clase OpenLayers.Style. Esta clase representa un Styled Layer Descriptor, SLD, en la linea de los definidos por el Open Geospatial Consortium. Los valores de atributos de estilo aceptados por las features vectoriales de OpenLayers son un subconjunto de los definidos por normas mencionadas anteriormente. Si no se especifica un estilo, OpenLayers asigna unos valores por defecto a estos atributos de estilo. La documentación de OpenLayers construye el objeto 'Style' a partir de un objeto con los valores por defecto, utilizando para ello el método 'extend' de la clase OpenLayers.Util. Este método lo que hace es copiar en el objeto destino todas las propiedades del objeto utilizado como origen (fuente). Nos devuelve el objeto destino modificado:

### Etiquetas en OpenLayers

Para mostrar etiquetas en mapas creados con OpenLayers podemos utilizar la clase OpenLayers.Feature.Vector. El constructor de Feature.Vector nos pide dos parámetros: una geometría y un estilo:

```
var feat = new OpenLayers.Feature.Vector(geometry, null, style);
```

(El segundo parámetro es un objeto con atributos opcionales para incorporar a la feature. En nuestro caso no lo vamos a utilizar) Para la geometría utilizaremos una instancia de la clase OpenLayers.Geometry.Point. Al constructor deberemos pasarle las coordenadas x,y del punto donde queramos situar la etiqueta:

```
var point = new OpenLayers.Geometry.Point(x,y);
```

Si las coordenadas provienen de objetos LonLat habrá que crearla de la siguiente manera:

```
var lonlat = new OpenLayers.LonLat(-3.74, 42.37);
var point = new OpenLayers.Geometry.Point(lonlat.lon,lonlat.lat);
```

Las features vectoriales tienen una propiedad denominada 'style' que es un objeto con las propiedades de visualización. Según la geometría que tenga la feature las propiedades se referirán a las características del grueso de linea, del radio del punto o, en nuestro caso, el estilo del texto. Las propiedades de estilo que puede tener una feature vectorial se detallan en la documentación de los symbolizer. La forma práctica de obtener un objeto estilo es utilizar el método 'extend' con un estilo por defecto y modificar el valor de las propiedades que queramos personalizar.

```
var pstyle = OpenLayers.Util.extend({},
 OpenLayers.Feature.Vector.style['default']
);
pstyle.graphic = false;
pstyle.labelX = "Hola Mundo";
pstyle.fontSize = "#000000";
pstyle.fontOpacity = 1;
pstyle.fontFamily="Arial";
pstyle.fontSize = "23";
pstyle.fontWeight = "bold";
```

La propiedad 'graphic' es un Boolean que indica si queremos representar el 'symbolizer' del punto. Poniendo el valor en 'false' sólo se visualizará la etiqueta. Si lo activamos con 'true' se verán la etiqueta y el punto(por defecto un círculo). En este caso disponemos de las propiedades 'labelXOffset' y 'labelYOffset' para situar el texto de la etiqueta respecto del símbolo del punto. Una vez creada la feature vectorial a partir de la geometría y el estilo habrá que añadirla a una capa vectorial que habremos creado previamente:

```
var layerVector = new OpenLayers.Layer.Vector("Vectorial");
var feat = new OpenLayers.Feature.Vector(
 point, null, labelst);
layerVector.addFeatures([feat]);
```

Hemos preparado un pequeño ejemplo en el que se utiliza un mapa cuya única capa es la capa vectorial que contiene la etiqueta. Esto se consigue pasando al mapa la propiedad 'allOverlays' con el valor 'true' y fijando una extensión para el mapa. Ejemplo de Etiqueta en OpenLayers

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>tl</title>
<link type="text/css" rel="stylesheet" href="http://www.bicimap.es/OpenLayers-2.10/theme/default/style.css"/>
<link rel="stylesheet" href="http://www.bicimap.es/OpenLayers-2.10/theme/default/ie6-style.css" type="text/css">
<script src="http://www.bicimap.es/OpenLayers-2.10/OpenLayers.js"></script>
<script>
var map,lay;
window.onload=function() {
var b = new OpenLayers.Bounds(-180,-80,180,80);
var map = new OpenLayers.Map("map",{
allOverlays: true,
maxExtent: b,
controls: []});
var lv = new OpenLayers.Layer.Vector("Vector");
var p = new OpenLayers.Geometry.Point(0,0);
var st = createLabelStyle("Hola Mundo");
var feat = new OpenLayers.Feature.Vector(p,null,st);
lv.addFeatures([feat]);
map.addLayer(lv);
map.zoomToMaxExtent();
```

```

}
function createLabelStyle(txt) {
var pstyle = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']
);
pstyle.graphic = false;
pstyle.label = txt;
pstyle.fontSize = "12px";
pstyle.fontColor = "#ff4500";
pstyle.fontFamily = "Verdana";
pstyle.fontWeight = "bold";
return pstyle;
};
</script>
</head>
<body>
<div id="map" style="width: 600px; height: 400px; border: solid 1px;"></div>
</body>
</html>

```

## Problemas en accesos locales

OpenLayers es una librería escrita en Javascript y tiene todas sus ventajas y sus limitaciones. Por seguridad no se permite que los scripts accedan a los ficheros del ordenador cliente, el que está ejecutando el navegador. Si que se permite acceder a ficheros en otros servidores.

Cuando realizamos una página html que accede a OpenLayers y muestra un mapa, siempre que los ficheros solicitados sean externos al ordenador cliente podremos ejecutarla directamente en el navegador y se verán los mapas solicitados. El problema viene cuando queremos utilizar ficheros de datos como pueden ser capas GML, capas de marcadores, etc. Si ejecutamos el html directamente en el navegador no podrá acceder a los ficheros de datos.

Para acceder a ficheros de datos debemos ejecutar nuestra página html a través de un servidor o bien que los ficheros de datos estén en un servidor. El esquema sería el siguiente:

En este esquema podemos utilizar el 'localhost' como servidor, pero en el navegador debemos ejecutar la página a través de él, esto es, con "http://localhost...".

## Añadir puntos dinámicamente

Vamos a comentar en este artículo un ejemplo sobre como añadir puntos a una capa vectorial a medida que se pulsa sobre el mapa. Para ello utilizaremos el control OpenLayers.Control.DrawFeature . El constructor de la clase tiene la siguiente expresión:

```
OpenLayers.Control.DrawFeature (
 layer,
 handler,
 options);
```

El primer parámetro es una instancia de una capa vectorial donde se irán guardando los puntos que vamos añadiendo. El segundo parámetro es un manejador del control, en este caso utilizaremos OpenLayers.Handler.Point. El tercer parámetro es opcional y permite incorporar opciones adicionales al control en el momento de su creación. En el ejemplo que acompaña el artículo, primero creamos una capa vectorial para alojar los puntos:

```
vectorLayer = new OpenLayers.Layer.Vector("Points");
```

Luego llamamos al constructor de OpenLayers.Control.DrawFeature para crear el control, al que le pasamos referencia de la capa vectorial, vectorLayer, y del manejador OpenLayers.Handler.Point. (Nosotros no utilizamos el tercer parámetro 'options'):

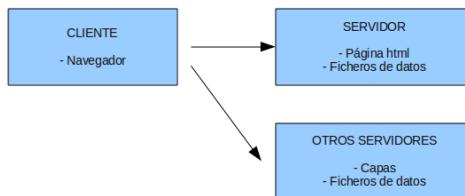
```
drawPoint=new OpenLayers.Control.DrawFeature (
 vectorLayer,
 OpenLayers.Handler.Point);
drawPoint.featureAdded = featAdded;
map.addControl(drawPoint);
```

El control OpenLayers.Control.DrawFeature tiene un atributo 'featureAdded' que se puede definir apuntando a un método que será llamado cada vez que se añada una feature, (un punto). En nuestro caso hemos definido una función que muestra en pantalla las coordenadas del último punto añadido.

Para activar el control, debemos llamar al método *'activate de la clase OpenLayers.Control'*. Esto provoca que cada vez que pulsemos en pantalla, se añada un punto a la capa vectorial 'vectorLayer'. En el ejemplo hemos habilitado unos botones para activar-desactivar la funcionalidad del control.

El ejemplo completo lo puedes ver en: [DynamicPoints](#)

```
<html>
<head>
<title>Puntos Dinámicos</title>
<meta charset="UTF-8">
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
```



Esquema de acceso a datos en OpenLayers

```
<script type="text/javascript">
var map, vectorLayer, controls, drawPoint;
function init(){
map = new OpenLayers.Map('map');
var wms = new OpenLayers.Layer.WMS("OpenLayers WMS", "http://labs.metacarta.com/wms/vmap0?", {layers: 'basic'});
vectorLayer = new OpenLayers.Layer.Vector("Points");
map.addLayers([wms, vectorLayer]);
map.addControl(new OpenLayers.Control.LayerSwitcher());
map.addControl(new OpenLayers.Control.mousePosition());
drawPoint=new OpenLayers.Control.DrawFeature(vectorLayer,OpenLayers.Handler.Point);
drawPoint.featureAdded = featAdded;
map.addControl(drawPoint);
map.setCenter(new OpenLayers.LonLat(0, 0), 3);
}
```

```

function featAdded() {
var el = document.getElementById("text");
el.value=drawPoint.handler.point.geometry.x+", "+drawPoint.handler.point.geometry.y;
}
function dibujar() {
drawPoint.activate();
}
function parar() {
drawPoint.deactivate();
}
</script>
</head>
<body onload="init()">
<div id="map" style="width: 480px; height: 300px;"></div>
<input type="button" value="Dibujar" onclick="dibujar()" />
<input type="button" value="Parar" onclick="parar()" />
Último punto: <input type="text" value="no set" id="text" width="30"/>
</body>
</html>
Código fuente de DynamicPoints

```

## Girar y mover features

Una feature consta de una geometría y un estilo. La geometría será una clase derivada de OpenLayers.Geometry. Estas clases tienen dos métodos 'move' y 'rotate' que permiten ajustar dinámicamente la posición de las features sobre el mapa. Las features tienen que haber sido creadas y añadidas a la capa vectorial previamente. Una vez llamadas las funciones 'move' y/o 'rotate' hay que llamar al método 'drawFeature()' de la capa para que se redibuje.

Al método 'rotate' hay que pasarle dos argumentos : un ángulo de rotación en grados (antihorario positivo) y un objeto OpenLayers.Geometry.Point que hará de centro de la rotación. rotate: function( angle: Float, center: OpenLayers.Geometry.Point)

El método 'move' acepta como parámetros el valor de los desplazamientos deseados en la dirección x e y: move: function( x: Float, y: Float)

El ejemplo de la documentación de OpenLayers ha servido para documentar el artículo y aplica los métodos aquí expuestos a geometrías *Point*, *LineString* y *LinearRing*.

*El método rotate no gira la feature sobre si misma, sino que realiza un giro de la feature respecto del centro de giro indicado. Si el centro de giro coincide con el centro de la feature, la feature no rota sobre si misma. Esto sucede cuando se utiliza una imagen en el estilo de una feature con geometría Point y se quiere girar sobre si misma, esto es, con centro de rotación en el centro de la feature, para orientar el icono en una determinada dirección. En este caso para girar la imagen habrá que actuar sobre la propiedad 'style.rotation' de la feature, asignándole el ángulo deseado.*

## Utilizar la proyección UTM en OpenLayers

En OpenLayers tenemos dos proyecciones definidas por defecto que son la EPSG:4326 (Coordenadas geográficas Datum WGS-84) y la EPSG:900913 (Spherical Mercator, habitualmente utilizada en los mapas de OpenStreetMap, Google y otros). Podemos incorporar otras proyecciones. Para ello tenemos que incluir en nuestro código la librería PROJ4JS que nos permitirá definir proyecciones y realizar las correspondientes transformaciones entre ellas. La librería Proj4JS la podemos descargar desde: <http://trac.osgeo.org/proj4js/wiki/Download>. Situaremos el fichero proj4js-combined.js en

un directorio accesible desde nuestra página web a la que añadiremos una sentencia del tipo:

```
<script src="lib/proj4js-combined.js"></script>
```

Ahora ya podemos utilizar en nuestro código Javascript la definición de proyecciones. Por ejemplo, para definir la proyección UTM zona 30 Norte, (EPSG:23030), debemos incluir en el código Javascript la siguiente sentencia:

```
Proj4js.defs["EPSG:23030"] = "+proj=utm +zone=30 +ellps=intl"+
 " +towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39 "+
 " +units=m +no_defs";
```

También podemos incluir una proyección concreta en nuestro código mediante la siguiente construcción:

```
<script
 src="http://spatialreference.org/ref/epsg/23030/proj4js/">
</script>
```

Ahora solamente nos queda utilizar las proyecciones definidas. Por ejemplo podríamos pasar de WGS84 a UTM-30N mediante la siguiente sentencia:

```
var wgs84Projection = new OpenLayers.Projection("EPSG:4326");
var utmProjection = new OpenLayers.Projection("EPSG:23030");
var defaultMapCenter = new OpenLayers.LonLat(-3.7, 40.985165)
 .transform(wgs84Projection, utmProjection);
```

*Las definiciones de las distintas proyecciones las podemos obtener en :*

<http://spatialreference.org/>

Hemos preparado un ejemplo completo en el que cargamos la capa del Mapa Topográfico Nacional Español Raster Escala 1:25.000 y la capa de las fotografías del Plan Nacional de Ortofotografía Aérea, PNOA. El resultado lo puedes ver en :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Manual OpenLayers</title>
<link type="text/css" rel="stylesheet" href="http://openlayers.org/dev/theme/default/style.css"/>
<style>
h1 {
line-height: 5px;
font-family: "Trebuchet MS", Helvetica, Arial, sans-serif;
font-size: 24px;
text-align: center;
text-shadow: 2px 2px 3px #333333;
color: #ff9900;
}
p {
line-height: 5px;
font-family: "Trebuchet MS", Helvetica, Arial, sans-serif;
font-size: 18px;
text-align: center;
text-shadow: 2px 2px 3px #333333;
color: #009900;
}
#logo {
float: left;
margin-top: 7px;
margin-left: 7px;
```

```

width: 80px;
}
#marco {
margin-left: auto;
margin-right: auto;
width: 800px;
height: 600px;
background: #86cd86;
padding: 5px;
}
#title {
width: 600px;
margin-left: auto;
margin-right: auto;
margin-top: 3px;
margin-bottom: 6px;
border: solid 1px #333333;
background: #f0f0f0;
}
#map {
width: 792px;
height: 450px;
border: solid 2px #333333;
}

```

</style>

```

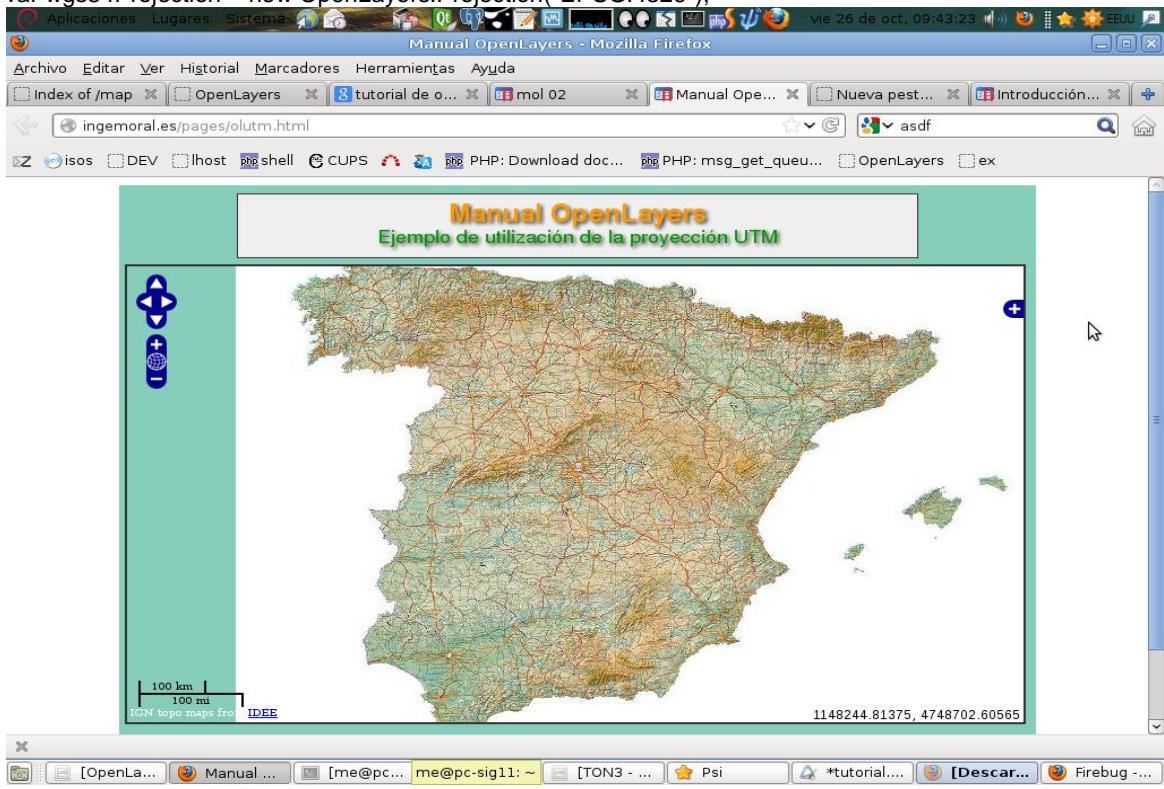
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="http://mercatorlab.com/proj4js/lib/proj4js-compressed.js"></script>

```

```

<script type="text/javascript">
//Projections
Proj4js.defs["EPSG:23030"] = "+proj=utm +zone=30 +ellps=intl"+
" +towgs84=-131,-100.3,-163.4,-1.244,-0.020,-1.144,9.39 +units=m +no_defs";
var wgs84Projection = new OpenLayers.Projection("EPSG:4326");

```



```

var sphericalMProjection = new OpenLayers.Projection("EPSG:900913");
var utmProjection = new OpenLayers.Projection("EPSG:23030");
// Mapa y capa vectorial

```

```

var map, vectorLayer;
// Centro mapa y zoom por defecto
//var defaultMapCenter = new OpenLayers.LonLat(2.828212, 41.985165)
var defaultMapCenter = new OpenLayers.LonLat(-3.7, 39.985165)
.transform(wgs84Projection, utmProjection);
var defaultMapZoom = 0;
//

window.onload = function() {

initMap();

};

function initMap() {
map = new OpenLayers.Map("map", {controls:[]});
// Capa base Mapa Topográfico Nacional Raster
addLayerMTNRaster(map);
// Capa base Plan Nacional Ortofotográfica Aérea
addLayerPNOA(map);
// Controles del mapa
map.addControl(new OpenLayers.Control.PanZoom());
map.addControl(new OpenLayers.Control.KeyboardDefaults());
var ctrlNav = new OpenLayers.Control.Navigation();
ctrlNav.zoomWheelEnabled = false;
map.addControl(ctrlNav);
map.addControl(new OpenLayers.Control.LayerSwitcher({title: 'Switch/add/remove layers'}));
map.addControl(new OpenLayers.Control.ScaleLine());
var ctrl = new OpenLayers.Control.mousePosition();
ctrl.displayProjection = utmProjection;
map.addControl(ctrl);
//
var ctrlAt = new OpenLayers.Control.Attribution();
map.addControl(ctrlAt);
ctrlAt.div.style.bottom="3px";
ctrlAt.div.style.left="3px";
ctrlAt.div.style.fontSize = "10px";
ctrlAt.div.style.color = "#ffffff";
// Centro y zoom
map.setCenter(defaultMapCenter, defaultMapZoom);
//map.zoomToMaxExtent();
}

function addLayerMTNRaster(mapref) {
mapref.projection = "EPSG:23030";
mapref.displayProjection = new OpenLayers.Projection("EPSG:4326");
mapref.maxExtent = new OpenLayers.Bounds(-100000, 3950000, 1150000, 4900000);
mapref.resolutions = [1800,900,450,225,120,50,25,10,4.5,3,2,1,0.5];
mapref.tileSize = new OpenLayers.Size(200, 200);
var ign = new OpenLayers.Layer.WMS("IGN topo",
"http://www.idee.es/wms/MTN-Raster/MTN-Raster", {layers: 'mtn_rasterizado'});
ign.attribution = 'IGN topo maps from IDEE';
map.addLayer(ign);
}

function addLayerPNOA(mapref) {
mapref.projection = "EPSG:23030";
mapref.displayProjection = new OpenLayers.Projection("EPSG:4326");
mapref.maxExtent = new OpenLayers.Bounds(-100000, 3950000, 1150000, 4900000);
mapref.resolutions = [1800,900,450,225,120,50,25,10,4.5,3,2,1,0.5];
mapref.tileSize = new OpenLayers.Size(200, 200);
var pnoa = new OpenLayers.Layer.WMS("PNOA aerial/topo",
"http://www.idee.es/wms/PNOA/PNOA", {layers: 'pnoa'},
{singleTile: true, ratio: 1});
pnoa.attribution = 'PNOA photos from IDEE';
mapref.addLayer(pnoa);
}

</script>
</head>

```

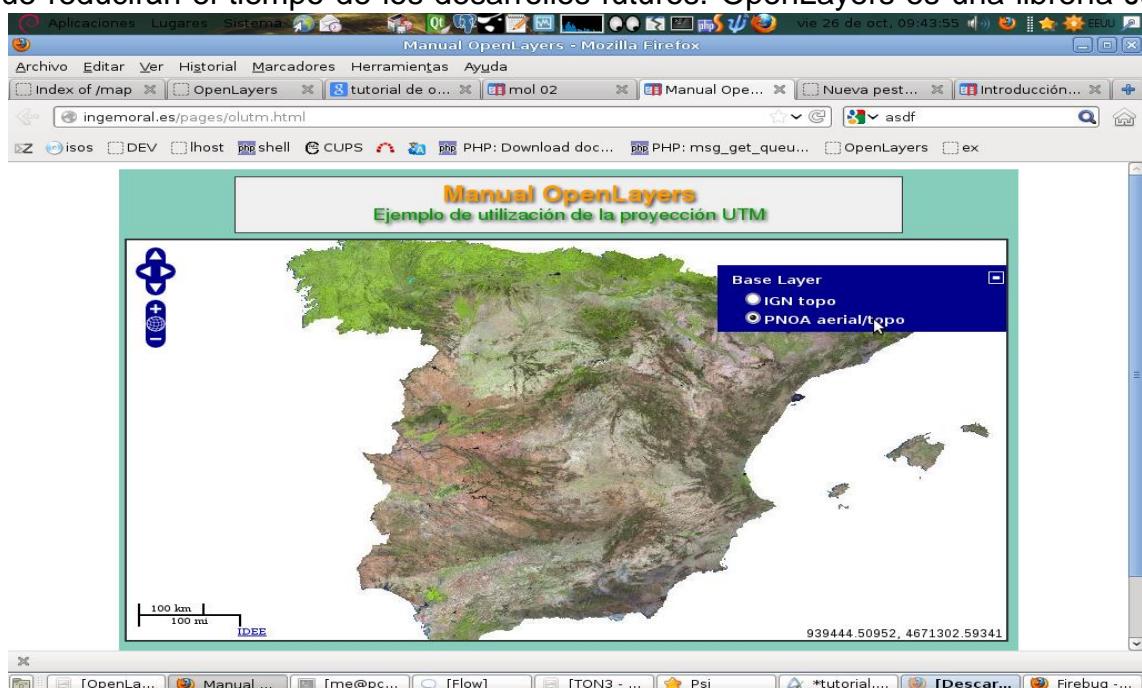
```

<body>
<div id="marco">
<div id="title">
<h1>Manual OpenLayers</h1>
<p>Ejemplo de utilizaciÃ³n de la proyecciÃ³n UTM</p>
</div>
<div id="map"></div>
</div>
</body>
</html>

```

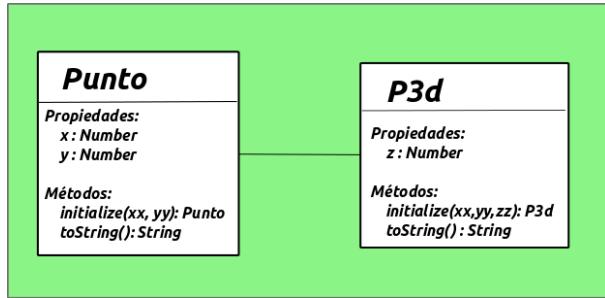
## Jerarquias de clases con OpenLayers

Programando en Javascript es posible adoptar un enfoque orientado a objetos. De esta forma, mediante el mecanismo de ir extendiendo las clases según una jerarquía determinada, vamos obteniendo objetos adaptados a nuestras necesidades particulares que reducirán el tiempo de los desarrollos futuros. OpenLayers es una librería Javascript



que utiliza esta aproximación, mediante una jerarquía de clases que dan servicio a las diferentes funcionalidades para las que se ha pensado el programa.

Podemos utilizar la arquitectura de clases de OpenLayers para crearnos una librería de clases propia que tenga objetos tales como 'miMapa', que incluye ya los controles y capas que suelo utilizar, o 'miFeature' que incorpora determinadas features utilizadas por mi programa con sus propias características de visualización, por ejemplo. Vamos a explicar en este artículo como crear una jerarquía propia de clases basada en la arquitectura proporcionada por OpenLayers. Para ello utilizaremos primero una jerarquía simple de objetos comunes como la indicada en el siguiente esquema:



En ella definimos las clases 'Punto' y 'P3d'. La clase Punto tiene como atributos las coordenadas x e y correspondientes a la representación de un punto en dos dimensiones. De ella deriva la clase 'P3d' que añade una tercera coordenada z. La clase 'P3d' hereda las propiedades y métodos de la clase 'Punto', algunos de los cuales los sobreescribe. Para implementar este esquema en nuestro programa Javascript+OpenLayers utilizaremos dos archivos:

- *Libreria js* : La hemos llamado 'clases1.js' y contiene las definiciones correspondientes a nuestra colección de clases.
- *Fichero html* : Lo hemos llamado 'clases1.html' y deberá incluir el link a la librería de OpenLayers y a nuestra librería de clases 'clases1.js'. Además, el fichero 'clases1.html', incorpora un pequeño script donde probamos la librería.

Vamos a analizar en primer lugar la librería 'clases1.js'. Es conveniente en primer lugar definir nuestro propio espacio de nombres, de forma que todas nuestras clases queden 'aisladas' del resto de librerías que utilice el programa. Hemos elegido llamar 'MisGeos' a nuestro espacio de nombres. Para definirlo se utiliza la siguiente construcción:

```
MisGeos = {};
```

A continuación definimos la clase 'MisGeos.Punto'. Para ello utilizaremos el objeto-función 'OpenLayers.Class' definido en la librería OpenLayers. Esta es la función encargada de crear nuevas clases y de gestionar el mecanismo de herencia. Para crear una nueva clase es necesario pasarle dos parámetros: La clase de la cual deriva nuestra nueva clase y, como segundo parámetro, un array asociativo con la definición de las nuevas propiedades y métodos que incorpora el nuevo objeto:

```
nuevaClase = OpenLayers.Class(
 ClasePadre,
 ArrayNuevasDefiniciones
);
```

El array del segundo parámetro tendrá la siguiente estructura:

```
ArrayNuevasDefiniciones= {
 prop1: valor1,
 prop2: valor2,
 metodo1: function() {...},
 metodo2: function(..) {...}
};
```

Vemos que cada elemento del Array es una definición de una propiedad o método y consta del nombre, los 'dos puntos' y la definición correspondiente. Lo veremos más claramente con el caso concreto de nuestra clase MisGeos.Punto:

```
MisGeos.Punto = OpenLayers.Class(MisGeos, {
 initialize: function(xx, yy) {
```

```

 this.x = xx;
 this.y = yy;
 },
 x : null,
 y : null,
 toString : function() {
 return "("+this.x+","+this.y+")";
 }
});

```

Vemos que el constructor de la clase es una función especial que se debe de llamar '*initialize*'. También hay que destacar cómo definimos las propiedades 'x' e 'y', a las que inicialmente asignamos un valor nulo, y cómo se procede a la asignación de sus valores. El constructor '*initialize*' recibe dos números 'xx' e 'yy' como parámetros y los asigna a las propiedades del objeto utilizando el 'this'. Este detalle es importante. Si no utilizamos 'this', los valores serían asignados a ciertas variables 'x' e 'y' locales del propio método '*initialize*'. El método '*toString()*' muestra el contenido de las propiedades 'x' e 'y' de la instancia, encerrados entre paréntesis y separados por una coma. Para crear una instancia de objeto de la clase MisGeos.Punto habrá que hacerlo de la siguiente manera:

```
var p1 = new MisGeos.Punto(-3.0,42.3);
```

Una vez creada la clase MisGeos.Punto, vamos a crear una clase 'MisGeos.P3d' que derive de la anterior y añada una tercera coordenada. La forma de hacerlo es la siguiente:

```

MisGeos.P3d = OpenLayers.Class(MisGeos.Punto, {
 initialize : function(xx, yy, zz) {
 MisGeos.Punto.prototype.initialize.apply(
 this, [xx, yy]);
 this.z = zz;
 },
 z : null,
 toString : function() {
 return "("+this.x+", "+this.y+", "+this.z+")";
 }
});

```

Aquí es importante darse cuenta de la manera de llamar al constructor de la clase base, MisGeos.Punto, desde el constructor de la clase 'MisGeos.P3d':

```

clasebase.prototype.initialize.apply (
 this,
 [parametros pasados al constructor]
);

```

Además podemos ver que se sobrescribe el método '*toString()*' de la clase base para adaptarlo a la nueva clase. Con esto queda creada la librería con nuestros dos tipos de objetos 'Punto' y 'P3d'. Para probar la librería utilizaremos un sencillo script, que se ha añadido en la página html, y que tiene la siguiente forma:

```

P1 = new MisGeos.Punto(-3.09, 45.78);
document.write ("Punto P1: " + P1.toString());
P2 = new MisGeos.P3d(10,20,30);
document.write ("P3d P2: " + P2.toString());

```

El código completo de los dos ficheros lo puedes ver:

```
MisGeos = {};
MisGeos.Punto = OpenLayers.Class(MisGeos, {
```

```

initialize: function(xx, yy) {
 this.x = xx;
 this.y = yy;
},
x : null,
y : null,
toString : function() {
 return "("+this.x+","+this.y+")";
}
});

MisGeos.P3d = OpenLayers.Class(MisGeos.Punto, {
initialize : function(xx, yy, zz) {
 MisGeos.Punto.prototype.initialize.apply(this, [xx,yy]);
 this.z = zz;
},
z : null,
toString : function() {
 return "("+this.x+", "+this.y+", "+this.z+")";
}
});

```

```

<!DOCTYPE html>
<html>
<head>
<meta charset= "UTF-8">
<title>pr1</title>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="clases1.js"></script>
</head>
<body>
<script>
P1 = new MisGeos.Punto(-3.09, 45.78);
document.write ("Punto P1: " + P1.toString()+"
");
P2 = new MisGeos.P3d(10,20,30);
document.write ("P3d P2: " + P2.toString()+"
");
</script>
</body>
</html>

```

## Forms que ejecutan Javascript

En ocasiones queremos leer datos del usuario en una página web y pasárselos a una rutina javascript como parámetros. Lo podemos hacer mediante un elemento `<form>` si tomamos algunas precauciones. Realizaremos un ejemplo en el que se le presenta al usuario un campo de texto para que teclee su nombre y un botón para validar. Una vez pulsado el botón se muestra el nombre tecleado en un elemento `<div>`.

Definimos un elemento 'form' que contendrá un campo tipo 'text' y un campo tipo 'button'. El 'text' tendrá definido un 'id' para poderlo identificar. El valor tecleado en el campo se puede recoger en el atributo 'value'. Para llamar a la rutina javascript necesitamos un campo tipo 'button' cuyo evento 'onclick' se define apuntando a la rutina que queremos ejecutar. Pasamos como parámetro el propio objeto 'form', de manera que la rutina pueda extraer los valores que le interesen. En nuestro caso hemos extraído el atributo 'value' del elemento 'textBox' que pertenece al 'form'. Este es el código completo de la página:

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavascriptForms</title>
<script type="text/javascript">
function validaNombre(sform) {
 var nom = sform.textBox.value;

```

```

 document.getElementById("nombre").innerHTML= "Nombre: " + nom;
 }
</script>
</head>
<body>
<form>
 <input type="text" id="textBox" size="40" value="Teclee su nombre"/>
 <input type="button" value="Validar"
 onclick="javascript:validaNombre(this.form)"/>
</form>
<div id="nombre">Nombre:</div>
</body>
</html>

```

Los detalles a destacar son:

- El elemento tipo 'text' lleva un 'id' definido (en este caso id='textBox')
- El elemento que llama a javascript es un elemento tipo 'button' a través del evento 'onclick'
- El 'form' se pasa como parámetro mediante la construcción 'this.form'
- El atributo 'value' de 'textBox' se extrae mediante 'sform.textBox.value'
- Otra forma de resolverlo sería sin pasar el 'form' como argumento, y recuperando el elemento 'textBox' directamente por su 'id':

```

<script type="text/javascript">
 function validaNombre() {
 var nom = document.getElementById("textBox").value;
 document.getElementById("nombre").innerHTML= "Nombre: " + nom;
 }
</script>

```

Las dos formas funcionan. La primera sería la forma antigua de hacerlo, mientras que esta segunda forma es la forma más actual de hacerlo.

## OpenStreetMap

El concepto de '*Open Street Map*' es como el de la Wikipedia pero aplicado al Mapa Mundi. La gente recopila información desde dispositivos GPS o desde imágenes satélite, la sube al portal, le añade etiquetas de nombres, topónimos, etc y la publica. El resultado es un Mapa Mundi que se puede ver en '<http://www.openstreetmap.org/>'.

*OpenStreetMap* es un producto Open Data, bajo una licencia Creative Commons, que permite utilizar el software o los datos, modificarlos, etc con la condición de mencionar el origen y en su caso de distribuir el producto bajo una licencia Open Data.

Cartografiar supone la mayor carga de trabajo en Open Street Map y tiene dos etapas. Primero, y la más divertida, es salir y capturar datos geográficos. Hay muchas maneras de hacerlo como son andando, en bicicleta o conduciendo. Cuando tengas algunos resultados, necesitarás agregarlos a la base de datos común de OSM utilizando uno de los editores. Esto es, cuando consigas agregar rutas, nombres de vías, buzones de correo o cualquier otra información que te gustaría incluir!.

Cuando los hagas, puedes sentarte, relajarte y ver los resultados de tu trabajo. ¡O bien salir fuera y hacer un mapa mayor y más completo!

También se puede intervenir en el proyecto colaborando en el desarrollo. Hay mucho trabajo todavía de desarrollo para hacer OpenStreetMap. Existen diferentes tipos de áreas de trabajo, y una variedad de idiomas y las tecnologías involucrados. También existe una sección española: <http://www.openstreetmap.es/>. A día de escribir este artículo hay inscritos 250.000 'mapeadores'. Cada día se añaden 25.000 <http://es.wikipedia.org/wiki/Km> nuevos de carreteras y caminos con un total de casi 34.000.000 km de viales, eso sin añadir otros tipos de datos (puntos de interés, edificaciones, etc.). El tamaño de la base de datos (llamada *planet.osm*) se sitúa por encima de los 160 gigabytes (6,1 GB con compresión bzip2), incrementándose diariamente en unos 10 megabytes de datos comprimidos.

## Añadir mapas estáticos

En este artículo vamos a explicar como añadir una imagen de un mapa de OpenStreetMap a una página HTML. Explicaremos como añadir una imagen estática y, en próximos artículos, explicaremos como añadirle capacidades de zoom, encuadre, marcadores, etc. Los datos que necesitamos son la Latitud y Longitud del punto central del mapa que queremos visualizar. Para obtenerlas podemos navegar por el mapa OSM y en la esquina inferior derecha podemos leer las coordenadas longitud, latitud del punto deseado. Una vez obtenidas la Latitud y Longitud lo único que hay que hacer es añadir un elemento *<img>* al código HTML de la página con el atributo 'src' apuntando a la dirección del mapa, como se ve a continuación:

```

```

Vemos que se trata de un elemento *imagen normal* con los atributos 'src', 'width', 'height' y 'alt' definidos. Veamos un poco en detalle los parámetros pasados en el atributo 'src' del elemento *<img>*:

- <http://ojw.dev.openstreetmap.org/StaticMap/?> : Es la URL del sitio Web de donde descargaremos el mapa. (La '?' separa la URL de los parámetros que vienen a continuación).
- lat=40.416&lon=-3.686 : Latitud y Longitud del punto central del mapa expresadas en grados sexagesimales. (El símbolo '&' separa cada parámetro del siguiente).
- z=11 : Es el nivel de zoom. Puede variar de 0 a 18. (OSM: Niveles de zoom)
- w=480&h=300: Son la anchura y la altura en pixels de la imagen solicitada.
- mode=Export&show=1: Parámetros necesarios para la petición. El parámetro show=1 indica que queremos ver la imagen del mapa.
- En la imagen que viene a continuación se puede ver el resultado de la petición realizada: Para una explicación mas detallada de los parámetros de la petición, así como para fabricarte el mapa a medida y solo tener que copiar y pegar el enlace puedes utilizar la siguiente dirección: Create a Map. En un próximo artículo explicaremos como añadir capacidades dinámicas al mapa, de forma que podamos hacer zoom en cualquier parte o añadir marcadores.

## Acceder desde OpenLayers

En este artículo trataremos la cuestión del acceso a los mapas de OpenStreetMap (OSM) utilizando la plataforma OpenLayers. OpenLayers ofrece una clase Layer especializada para el acceso a OSM: la clase *OpenLayers.Layer.OSM*. Si nos hemos descargado los códigos fuente de la librería, podremos encontrar el código de la clase

*OpenLayers.Layer.OSM en el archivo '/lib/OpenLayers/Layer/XYZ.js'. Reproducimos a continuación el código:*

```
* Class: OpenLayers.Layer.OSM
* A class to access OpenStreetMap tiles. By default, uses the OpenStreetMap
* hosted tile.openstreetmap.org 'Mapnik' tileset. If you wish to use
* tiles@home / osmarender layer instead, you can pass a layer like:
*
* new OpenLayers.Layer.OSM("t@h",
* "http://tah.openstreetmap.org/Tiles/tile/${z}/${x}/${y}.png");
*
* This layer defaults to Spherical Mercator.
*/
OpenLayers.Layer.OSM = OpenLayers.Class(OpenLayers.Layer.XYZ, {
 name: "OpenStreetMap",
 attribution: "Data CC-By-SA by OpenStreetMap",
 sphericalMercator: true,
 url: 'http://tile.openstreetmap.org/${z}/${x}/${y}.png',
 CLASS_NAME: "OpenLayers.Layer.OSM"
});
```

*El sistema de referencia de la capa OSM es el Spherical Mercator, que corresponde al código "EPSG:900913", por lo que si partimos de puntos conocidos en otro sistema de referencia habrá que hacer la transformación correspondiente. Esto se puede hacer mediante el método *transform()* de la clase *OpenLayers.LonLat*. En el ejemplo que acompaña este artículo queremos centrar el mapa en un punto de coordenadas lon=-3.69357, lat=40.41062 expresadas en el sistema de referencia WGS84, que le corresponde el código "EPSG:4326". Podemos hacer la transformación de la siguiente manera:*

```
var projWGS84 = new OpenLayers.Projection("EPSG:4326");
var projSphericalMercator = new OpenLayers.Projection("EPSG:900913");

var centerWGS84=new OpenLayers.LonLat(-3.69357,40.41062);
// transform from WGS 1984 to Spherical Mercator Projection;
var centerOSM =centerWGS84.transform(projWGS84, projSphericalMercator);
```

Donde hemos definido dos objetos *OpenLayers.Projection*, un punto de coordenadas conocidas en el Datum WGS84 y un punto de coordenadas en el sistema Spherical Mercator que se obtiene al aplicar el método *transform* de la clase *LonLat*. Con ésto resuelto, el resto de la función para construir el mapa es sencillo. Reproducimos a continuación el listado de una sencilla página web para mostrar un mapa OSM:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=ISO-8859-1" />
<title>Test OpenStreetMap</title>

<style type="text/css">
 h3 {
 font: 100% 'Lucida Grande', Verdana, Arial, sans-serif;
 font-size: 110%;
 color: #003a6b;
 background-color: transparent;
 margin: 0; padding-top: 0.5em;
 margin-bottom: 0.5em;
 border-bottom: 1px solid #fcb100;
 }
 .cabecera {
```

```

 position: absolute;
 top: 10px; left: 10px;
 background: #00a000; color:f0f0f0;
 margin: 10px; padding: 10px;
 }
 .map {
 position: absolute;
 left: 10px; top: 100px;
 width: 640px; height: 400px;
 border: solid 2px #a0a0a0;
 background-color:#ffffff;
 color: white;
 padding: 3px 3px 3px 3px;
 }
</style>

<script src="http://www.openlayers.org/api/OpenLayers.js"></script>

<script type="text/javascript">
 var map, osmLayer, centerWGS84, centerOSM;
 var projWGS84, projSphericalMercator

 function init() {
 projWGS84 =
 new OpenLayers.Projection("EPSG:4326");
 projSphericalMercator =
 new OpenLayers.Projection("EPSG:900913");
 centerWGS84 =
 new OpenLayers.LonLat(-3.69357,40.41062);
 // transform from WGS 1984 to Spherical Mercator Projection;
 centerOSM =
 centerWGS84.transform(projWGS84, projSphericalMercator);

 map = new OpenLayers.Map("mapa");
 osmLayer = new OpenLayers.Layer.OSM();
 map.addLayer(osmLayer);
 map.setCenter(centerOSM, 5);
 }
</script>

</head>

<body onload="init()">
 <div class="cabecera" >
 <h3>Prueba de acceso a OpenStreetMap</h3>
 </div>
 <div class="map" id="mapa" ></div>
</body>
</html>

```

La capa OpenLayers.Layer.OSM por defecto accede al juego de imágenes 'Mapnik'. Si queremos acceder a las imágenes de OsmaRender debemos hacer la siguiente llamada al constructor de la capa:

```
osmLayer = new OpenLayers.Layer.OSM("t@h",
 "http://tah.openstreetmap.org/Tiles/tile/${z}/${x}/${y}.png");
```

El resultado de ambos tipos de mapa puede verse en los siguientes enlaces:

## Acceder desde OpenLayers

*En este artículo trataremos la cuestión del acceso a los mapas de OpenStreetMap (OSM) utilizando la plataforma OpenLayers. OpenLayers ofrece una clase Layer especializada para el acceso a OSM: la clase OpenLayers.Layer.OSM. Si nos hemos descargado los códigos fuente de la librería, podremos encontrar el código de la clase OpenLayers.Layer.OSM en el archivo '/lib/OpenLayers/Layer/XYZ.js'. Reproducimos a continuación el código:*

```

* Class: OpenLayers.Layer.OSM
* A class to access OpenStreetMap tiles. By default, uses the OpenStreetMap
* hosted tile.openstreetmap.org 'Mapnik' tileset. If you wish to use
* tiles@home / osmarender layer instead, you can pass a layer like:
*
* new OpenLayers.Layer.OSM("t@h",
* "http://tah.openstreetmap.org/Tiles/tile/${z}/${x}/${y}.png");
*
* This layer defaults to Spherical Mercator.
*/
OpenLayers.Layer.OSM = OpenLayers.Class(OpenLayers.Layer.XYZ, {
 name: "OpenStreetMap",
 attribution: "Data CC-BY-SA by OpenStreetMap",
 sphericalMercator: true,
 url: 'http://tile.openstreetmap.org/${z}/${x}/${y}.png',
 CLASS_NAME: "OpenLayers.Layer.OSM"
});

```

*El sistema de referencia de la capa OSM es el Spherical Mercator, que corresponde al código "EPSG:900913", por lo que si partimos de puntos conocidos en otro sistema de referencia habrá que hacer la transformación correspondiente. Esto se puede hacer mediante el método *transform()* de la clase *OpenLayers.LonLat*. En el ejemplo que acompaña este artículo queremos centrar el mapa en un punto de coordenadas lon=-3.69357, lat=40.41062 expresadas en el sistema de referencia WGS84, que le corresponde el código "EPSG:4326". Podemos hacer la transformación de la siguiente manera:*

```
var projWGS84 = new OpenLayers.Projection("EPSG:4326");
var projSphericalMercator = new OpenLayers.Projection("EPSG:900913");
```

```
var centerWGS84=new OpenLayers.LonLat(-3.69357,40.41062);
// transform from WGS 1984 to Spherical Mercator Projection;
```

var centerOSM =centerWGS84.transform(projWGS84, projSphericalMercator);

Donde hemos definido dos objetos *OpenLayers.Projection*, un punto de coordenadas conocidas en el Datum WGS84 y un punto de coordenadas en el sistema Spherical Mercator que se obtiene al aplicar el método *transform* de la clase *LonLat*. Con ésto resuelto, el resto de la función para construir el mapa es sencillo. Reproducimos a continuación el listado de una sencilla página web para mostrar un mapa OSM:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=ISO-8859-1" />
<title>Test OpenStreetMap</title>

<style type="text/css">
 h3 {
 font: 100% 'Lucida Grande', Verdana, Arial, sans-serif;
 font-size: 110%;
 color: #003a6b;
 background-color: transparent;
 margin: 0; padding-top: 0.5em;
 margin-bottom: 0.5em;
 border-bottom: 1px solid #fcb100;
 }
 .cabecera {
 position: absolute;
 top: 10px; left: 10px;
 background: #00a000; color:f0f0f0;
```

```

 margin: 10px; padding: 10px;
 }
 .map {
 position: absolute;
 left: 10px; top: 100px;
 width: 640px; height: 400px;
 border: solid 2px #a0a0a0;
 background-color:#ffffff;
 color: white;
 padding: 3px 3px 3px 3px;
 }
</style>

<script src="http://www.openlayers.org/api/OpenLayers.js"></script>

<script type="text/javascript">
 var map, osmLayer, centerWGS84, centerOSM;
 var projWGS84, projSphericalMercator

 function init() {
 projWGS84 =
 new OpenLayers.Projection("EPSG:4326");
 projSphericalMercator =
 new OpenLayers.Projection("EPSG:900913");
 centerWGS84 =
 new OpenLayers.LonLat(-3.69357,40.41062);
 // transform from WGS 1984 to Spherical Mercator Projection;
 centerOSM =
 centerWGS84.transform(projWGS84, projSphericalMercator);

 map = new OpenLayers.Map("mapa");
 osmLayer = new OpenLayers.Layer.OSM();
 map.addLayer(osmLayer);
 map.setCenter(centerOSM, 5);
 }
</script>

</head>

<body onload="init()">
 <div class="cabecera" >
 <h3>Prueba de acceso a OpenStreetMap</h3>
 </div>
 <div class="map" id="mapa" ></div>
</body>
</html>

```

*La capa OpenLayers.Layer.OSM por defecto accede al juego de imágenes 'Mapnik'. Si queremos acceder a las imágenes de Osmarender debemos hacer la siguiente llamada al constructor de la capa:*

```

osmLayer = new OpenLayers.Layer.OSM("t@h",
 "http://tah.openstreetmap.org/Tile/tile/${z}/${x}/${y}.png");

```

*El resultado de ambos tipos de mapa puede verse en los siguientes enlaces:*

### Acceder a los distintos mapas

*El acceso a OpenStreetMap está documentado en OpenLayers para hacerlo a través de la clase OpenLayers.Layer.OSM. Podemos acceder a las capas Mapnik y Osmarender cambiando la url de conexión de la siguiente forma:*

```

// La capa Mapnik es la que carga por defecto.
// La url es: http://tile.openstreetmap.org/${z}/${x}/${y}.png
var layerMapnik = new OpenLayers.Layer.OSM("Mapnik");

// Para cargar Osmarender hay que especificar la url
var layerOsmarender = new OpenLayers.Layer.OSM(

```

```

 "Osmarender",
 "http://tah.openstreetmap.org/Tiles/tile/${z}/${x}/${y}.png"
);

```

Es posible acceder de una forma más sencilla a las capas Mapnik y Osmarender y además acceder a CycleMap. Para ello debemos incluir en la cabecera del documento un enlace al siguiente script:

```
<script src="http://openstreetmap.org/openlayers/OpenStreetMap.js"></script>
```

Este script nos añade las clases OpenLayers.Layer.OSM.Mapnik, OpenLayers.Layer.OSM.Osmarender y OpenLayers.Layer.OSM.CycleMap. Su utilización es sencilla. A continuación reproducimos el código de una pagina html sencilla que da la posibilidad de elegir entre las distintas capas de OpenStreetMap:

```

<!DOCTYPE html>
<html>
<head>
<title>OSM Layers</title>
<!-- Hoja de estilo del programa -->
<style type="text/css">
#mapOSMLayers {
 width: 100%;
 height: 600px;
 padding: 0;
 margin: 0;
}
</style>
<!-- Cargar la librería OpenLayers -->
<script src="http://openlayers.org/api/OpenLayers.js"></script>
<!-- Cargar las clases OpenLayers.Layer.OSM.Mapnik,
 OpenLayers.Layer.Osmarender y OpenLayers.Layer.OSM.CycleMap -->
<script src="http://openstreetmap.org/openlayers/OpenStreetMap.js"></script>

<!-- Script del programa -->
<script type="text/javascript">
var map;
function init() {
 map = new OpenLayers.Map("mapOSMLayers");
 map.addControl(new OpenLayers.Control.LayerSwitcher());

 var layMapnik = new OpenLayers.Layer.OSM.Mapnik("Mapnik");
 map.addLayer(layMapnik);

 var layOsmarender = new OpenLayers.Layer.OSM.Osmarender("Osmarender");
 map.addLayer(layOsmarender);

 var layCycleMap = new OpenLayers.Layer.OSM.CycleMap("CycleMap");
 map.addLayer(layCycleMap);

 var center = new OpenLayers.LonLat(-0.11609, 51.5043);
 var centerOSM = center.transform(
 new OpenLayers.Projection("EPSG:4326"),
 new OpenLayers.Projection("EPSG:900913")
);

 map.setCenter(centerOSM, 10);
 map.setBaseLayer(layCycleMap);
}
</script>
</head>
<body onload="init()">
 <div id="mapOSMLayers"></div>
</body>
</html>

```

## 6.9 Un enfoque diferente basado en HTML5

En 1997 W3C<sup>30</sup> publica HTML4 la ultima versión hasta hace unos años del lenguaje de marcas para desarrollar aplicaciones web, un año mas tarde en 1998 publican la especificación 2 de CSS denominado CSS2, un lenguaje para describir los estilos de las aplicaciones web. Desde aquellos tiempos hasta hoy muchas de las aplicaciones web han evolucionado a grandes niveles de interacción, tratando de semejarse a las convencionales aplicaciones de escritorio. Estas evoluciones crearon la necesidad de integrar a la web plugins como Adobe Flash, Silverlight, VLC video y demás.

En junio de 2004 representantes de la comunidad de la web semántica y el W3C se reúnen para discutir estándares sobre la web, comienza así a trabajarse sobre las primeras especificaciones de HTML5, un lenguaje con nuevos elementos y API<sup>31</sup> que entre otras ventajas eliminan las dependencias a plugins de terceros como los mencionados anteriormente, lográndose aplicaciones mas usables y de mayor interoperabilidad.

Hoy día, muchas de las aplicaciones que están ganando popularidad son los sistemas de información geográfica web. Así en 2005 Google proporciona un API de Javascript para su servicio de mapas Google Maps, lo cual facilitó la integración de mapas interactivos en la web. Inspirado en esta innovación, en el año 2006 se libera OpenLayers<sup>32</sup> como una biblioteca elaborada completamente en javascript y siguiendo las especificaciones de HTML4 para integrar mapas interactivos en la web.

En este trabajo se integran algunos de los elementos de HTML5 a OpenLayers con el objetivo de brindar mayores posibilidades a esta herramienta. Entre estos se encuentran el elemento Canvas como elemento fundamental de repintado de mapas, los WW<sup>33</sup> para procesar javascript en hilos separados, almacenamiento local para optimizar el repintado de mapas y poder trabajar offline.

En este trabajo se evalúa la representación de imágenes Raster<sup>34</sup> mediante Canvas, lo cual permite la manipulación de bits haciendo posible muchas operaciones de procesado.

Los mapas creados con *OpenLayers* presentan interfaces muy interactivas que usan AJAX para hacer peticiones asincrónicas al servidor de mapas, en muchos casos hay gran procesamiento de datos en ejecución y aumento considerable de peticiones al servidor.

### 6.9.1 Elementos HTML5

#### 6.9.1.1 El elemento Canvas

El Canvas es un elemento *HTML* que puede ser usado por javascript entre otras cosas para dibujar imágenes. En el año 2004 fue introducido por Apple en su diseño de motor

<sup>30</sup> <http://en.wikipedia.org/wiki/W3C>

<sup>31</sup> <http://en.wikipedia.org/wiki/Api>

<sup>32</sup> <http://openlayers.org>

<sup>33</sup> WW acrónimo de Web Worker, [http://en.wikipedia.org/wiki/Web\\_worker](http://en.wikipedia.org/wiki/Web_worker)

<sup>34</sup> <http://en.wikipedia.org/wiki/Raster>

Web Kit<sup>35</sup> HTML y en la actualidad esta normalizado por el WHATWG<sup>36</sup> en la especificación de HML5. Así para este año 2012, el Canvas cuenta con soporte en estos navegadores:

	Firefox 12	Chrome 18	Internet Explorer 10
<b>Canvas</b>	<b>345</b> 9 bonus points	<b>400</b> 13 bonus points	<b>316</b> 6 bonus points
canvas element »	Yes ✓	Yes ✓	Yes ✓
2D context »	Yes ✓	Yes ✓	Yes ✓
Text »	Yes ✓	Yes ✓	Yes ✓

Ilustración 79 Compatibilidad del elemento Canvas en los principales navegadores

La representación de elementos o imágenes en el Canvas se hace a través de javascript. El objeto context de dos dimensiones del Canvas implementa la interfaz CanvasRenderingContext2D que ofrece las funciones de dibujo y las propiedades de los estilos de dibujo. El elemento Canvas no es más que una imagen sobre la cual podemos operar, de ahí que podemos acceder a los valores de cada pixel a través del objeto CanvasPixelArray, un arreglo con los valores RGBA de todos los pixeles del Canvas.

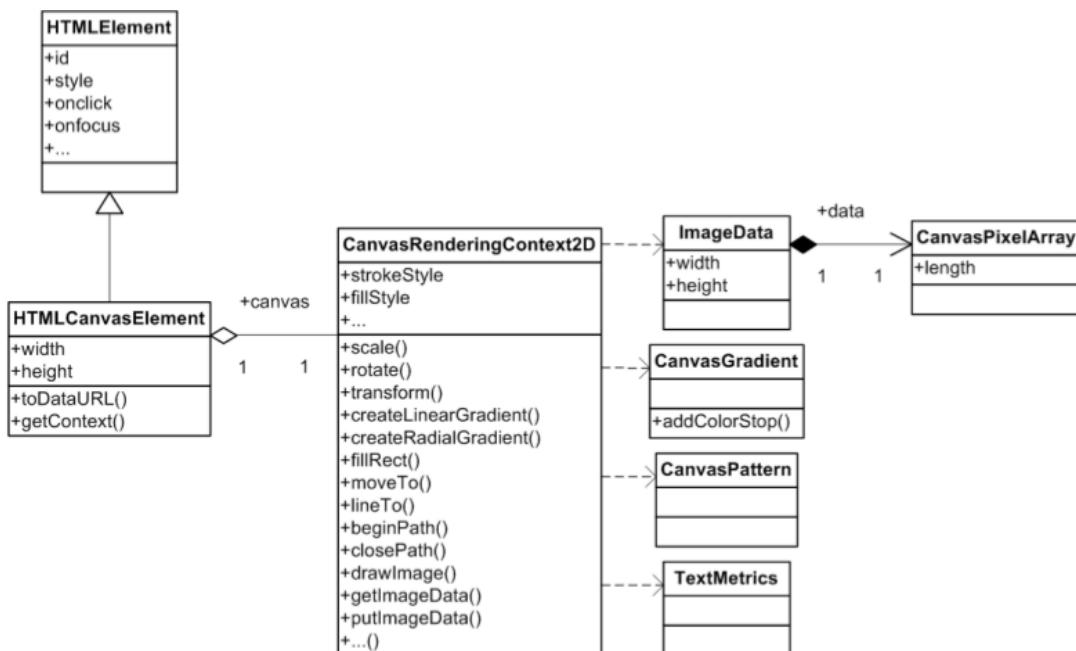


Ilustración 80 Diagrama de clases fundamentales para el API del elemento Canvas

### 6.9.1.2 WebWorker

La especificación de los WW define un API para la ejecución de script en segundo plano (hilos de ejecución diferentes) en la web, y evitar el bloqueo de la interfaz de usuario en tareas de operaciones largas. Así un WW se ejecuta en su propio hilo, el concepto de simultaneidad es diferente a otros lenguajes de programación puesto que javascript no tiene construcciones del lenguaje para el acceso sincronizado de los recursos como

<sup>35</sup> [http://en.wikipedia.org/wiki/Web\\_Kit](http://en.wikipedia.org/wiki/Web_Kit)

<sup>36</sup> <http://www.whatwg.org/>

variables y datos, por tanto el intercambio entre los WW y el script principal es a través de mensajes, es preciso señalar que por esta razón no puede accederse directamente al DOM o al objeto window desde un WW.

	Firefox 12	Chrome 18	Internet Explorer 10
<b>345</b>	<b>400</b>	<b>316</b>	
9 bonus points	13 bonus points	6 bonus points	
Workers	10	15	10
Web Workers >>	Yes ✓	Yes ✓	Yes ✓
Shared Workers >>	No ✗	Yes ✓	No ✗

**Ilustración 81** Compatibilidad de los webworker en los principales navegadores

#### 6.9.1.3 DOM Storage

El almacenamiento DOM (DOM Storage) es el nombre dado al conjunto de características relacionadas con el almacenamiento introducidas en la especificación de aplicaciones web 1.0 y ahora detalladas por separado en su propia especificación W3C Web Storage. El almacenamiento DOM está diseñado para facilitar una forma amplia, segura y sencilla para almacenar información alternativa a las cookies.

El mecanismo de almacenamiento DOM es el medio a través del cual pares de clave/valor pueden ser almacenadas de forma segura para ser recuperadas y utilizadas más adelante. La meta de este añadido es suministrar un método exhaustivo a través del cual puedan construirse aplicaciones interactivas (incluyendo características avanzadas tales como ser capaces de trabajar "sin conexión" durante largos períodos de tiempo).

Actualmente sólo los navegadores basados en Mozilla, Internet Explorer 8 y Safari (otros navegadores basados en webkit, como Google Chrome, no) proporcionan una implementación funcional de la especificación del almacenamiento DOM. Sin embargo, las versiones anteriores a Internet Explorer 8 poseen una característica similar llamada "userData behavior" que permite conservar datos entre múltiples sesiones.

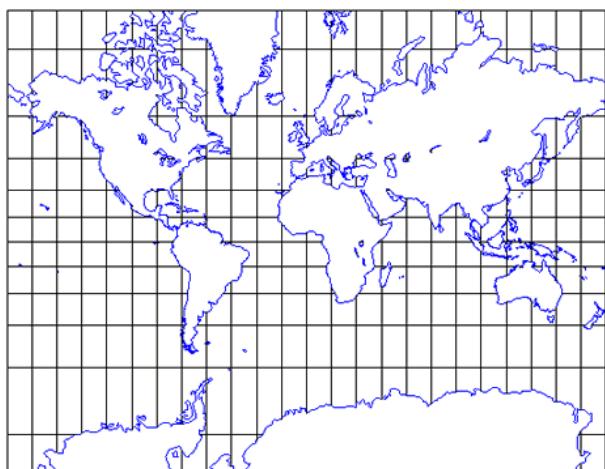
El almacenamiento DOM es útil ya que ningún navegador dispone de buenos métodos para conservar cantidades razonables de datos durante un periodo de tiempo. Las cookies de los navegadores tienen una capacidad limitada y no implementan una forma de organizar datos persistentes y otros métodos (tales como almacenamiento local de Flash) necesitan un plugin externo.<sup>37</sup>

#### 6.9.2 Representación de datos espaciales.

Raster y Vectorial: En general, los datos geográficos son un modelo digitalizado de fenómenos reales, una abstracción simplificada de la compleja realidad. Tradicionalmente hay dos modelos de datos fundamentales: Raster y Vectorial. Ambos tienen en común que hacen referencia a los objetos de la superficie de la tierra utilizando un sistema de coordenadas geográficas. Un sistema común para identificar sistemas de coordenadas y proyecciones de mapas es el código EPSG, así ejemplos de códigos ampliamente usados

<sup>37</sup> <https://developer.mozilla.org/es/DOM/Almacenamiento>

son el EPGS: 4326 que utiliza el *datum*<sup>38</sup> WGS84, y EPGS: 900913 que utiliza la proyección esférica de Mercator.



**Ilustración 82** Proyección de Mercator

En la mayoría de los SIG web los datos Raster son mostrados como imágenes en formato de ficheros solo almacenan la información del color como PNG o JPEG puesto que los navegadores no soportan interpretación de estos modelos. Los datos Raster en SIG web son a menudo imágenes satelitales o aéreas que son publicados a través de servicios web de mapas (WMS). Estos servicios permiten personalizar la imagen del mapa solicitado, aceptando parámetros como tamaño de la imagen, proyección, fuente de datos y demás. Esta flexibilidad es también un inconveniente cuando se utiliza un servicio WMS por un número elevado de usuarios. Debido a que cada petición WMS genera una imagen, muchas solicitudes simultáneas dan lugar a un cuello de botella de rendimiento. Para esto muchas aplicaciones adoptan enfoques diferentes como servir servicios de cache de mapas.

### 6.9.3 Clases capas de OpenLayers

Una de las características claves de los mapas es mostrar los datos con diferentes sistemas de coordenadas y proyecciones en un mapa. Un mapa consta de una o mas capas. Una capa puede tener una o más fuentes de datos y describe como los visualiza. Cada mapa utiliza exactamente una proyección, pero puede contar con capas con diferentes proyecciones que son re proyectadas al vuelo a la proyección base en el proceso de visualización.

#### 6.9.3.1 Tipos de Capas en OpenLayers.

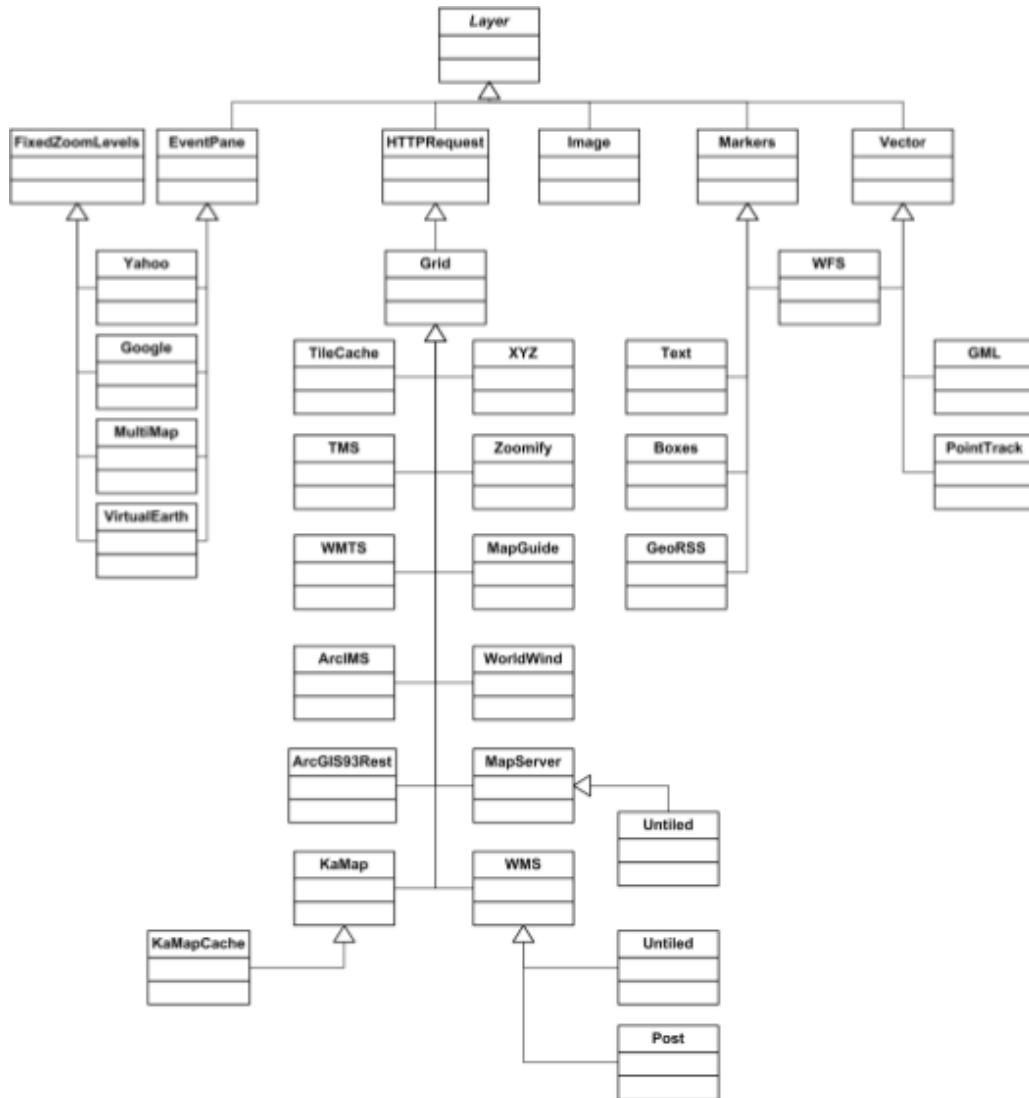
OpenLayers tiene dos tipos de capas fundamentales, la capa base y las superpuestas. OpenLayers soporta una amplia gama de tipos de capas diferentes, las tres categorías más relevantes son EventPane, HTTPREQUEST/Grid, y Vector.

La clase EventPane proporciona una manera de integrar los datos del mapa entre OpenLayers y un API de un tercero como Google Maps, Bing Maps, esta clase básicamente es un proxy para traducir entre OpenLayers y un API en particular.

<sup>38</sup> [http://en.wikipedia.org/wiki/Datum\\_\(geodesy\)](http://en.wikipedia.org/wiki/Datum_(geodesy))

La característica principal de una capa Grid es que muestra los datos en imágenes que se dividen por cuadrículas. Cada imagen en cada cuadrícula se obtiene a través de una petición al servidor de mapas. Las clases derivadas de la clase Grid implementan diversas especificaciones de consumo de servicios de Mapas como WMS WMTS, TileCache, Mapserver, entre otras.

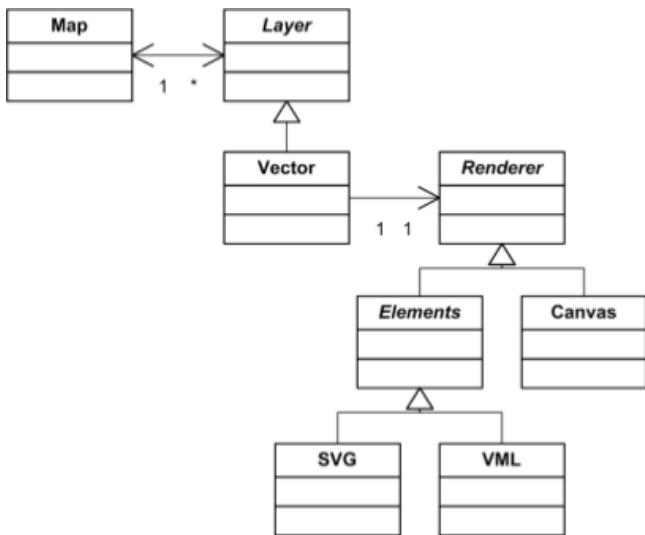
La clase Vector se utiliza para representar geometrías de elementos vectoriales, también provee de interfaces para interactuar con estas geometrías y posee compatibilidad con un gran número de formatos como GML, GeoJSON, KML, WKT entre otros.



**Ilustración 83** Diagrama de clases para las capas en OpenLayers

#### 6.9.3.2 Renderizado de datos vectoriales

En OpenLayers, los datos vectoriales están representados mediante la clase Vector. Cada capa vector tiene asociado un procesador de dibujo (renderer) que pinta los diferentes objetos que contiene capa. En la actualidad OpenLayers tiene tres procesadores de dibujo según las diferentes tecnologías a usar, estos son los dos formatos vectoriales SVG y VML y el Canvas de HTML. Que renderer se utiliza en una aplicación de OpenLayers depende que tecnología sea compatible con el navegador. Por defecto usa SVG si esta disponible, seguido de VML y por ultimo canvas.



**Ilustración 84** Diagrama de clases para el dibujo en OpenLayers

En OpenLayers cuando una capa vectorial se dibuja, el renderer de SVG inserta un elemento SVG en el div contenedor de la capa. El renderer SVG crea también dos SVG uno para la geometría y otro para el texto de la misma.

En renderer Canvas para la clase Vector es parte de Open Layers desde el año 2008, Este renderer crea un HTMLCanvasElement y lo inserta en el elemento div de la capa vector. Al igual que el renderer SVG el Canvas no actualiza la vista durante el arrastre del mapa por razones de rendimiento.

#### 6.9.3.3 Renderizado de datos raster

La clase Grid en OpenLayers. Es la clase padre de todas las capas Raster que puedan añadirse al mapa. La información en estas capas se presenta en cuadrículas. Esta clase Grid tiene una matriz de elementos de tipo Tile.Image que es responsable de mostrar la imagen de una cuadrícula. La clase Grid tiene una propiedad singleTile que si se establece a true, solo posee un elemento Tile.Image, o sea, la matriz de elementos Tile.Image solo posee una celda.

Del mismo modo que en la clase Vector, siempre que hay cambios en la extensión del mapa, el método moveTo es llamado, posterior a esto la matriz de elementos de la clase Grid se vuelve a recalcular para eliminar las filas y columnas que ya no son necesarias y para adicionar aquellas que se están dentro del área visible del mapa. Si el nivel de zoom ha cambiado la matriz entera debe ser recalculada y los tiles se cargan asincrónicamente en forma de espiral.

##### 6.9.3.3.1 Implementación

Apoyado sobre la base de los nuevos elementos de HTML5 se pueden realizar algunas modificaciones a la clase Grid, para usar el Canvas en la generación de las cuadrículas. En este caso la idea es utilizar un elemento Canvas por cada capa Grid que contenga el mapa. El procesamiento de generación de la matriz es similar a la implementación de OpenLayers, solo cambia la forma de procesado de Imágenes HTML a Canvas.

Para esto se implementa una nueva clase llamada CanvasTile, que es una subclase OpenLayers.Tile. Cuando cambia la extensión del mapa, la capa Grid crea elementos de

tipo canvasTile para formar sus cuadrículas y se dibujan asincrónicamente en el Canvas respectivo de esta capa.

```
addTile: function(bounds, position){
 var tile;
 if(this.useCanvas){
 tile = new OpenLayers.Tile.CanvasTile(this, position, bounds, null, this.tileSize);
 } else {
 tile = new OpenLayers.Tile.Image(this, position, bounds, null, this.tileSize);
 }
 tile.url = OpenLayers.Layer.proxyURL(this.url);
 return tile;
},
```

Para la petición de la imagen es opcional usar un proxy (OpenLayers.Layer.proxyURL) y evitar errores de seguridad en el posterior tratamiento de estas imágenes como aclararlas, cambiar tonos, detectar bordes y demás que requieren el control sobre los píxeles de la misma. Por defecto se usa un proxy en la aplicación ejemplo.

#### 6.9.3.3.2 Análisis de rendimiento

En esta sección se compara los diferentes enfoques de repintado de las capas Grid para el caso de prueba generar Capa Grid se centra el mapa a diferentes niveles de zoom aleatorios. Para esto se usa una capa de WMS que usa una cache de mapas, de modo que los problemas de la red influyan lo menos posible en el resultado del experimento. Se usan diferentes tamaños de mapas, de modo que se tienen imágenes de 196x267, 485x379, 727x545, 1034x594, 1340x847, 1652x1107 y 2384x1129, el buffer de la capa esta en cero, solo cargando las cuadrículas visibles. Las pruebas son realizadas en un PC con sistema Operativo Ubuntu12.04, 2 GB memoria RAM, procesador Intel Dual Core 2.70 GHz.

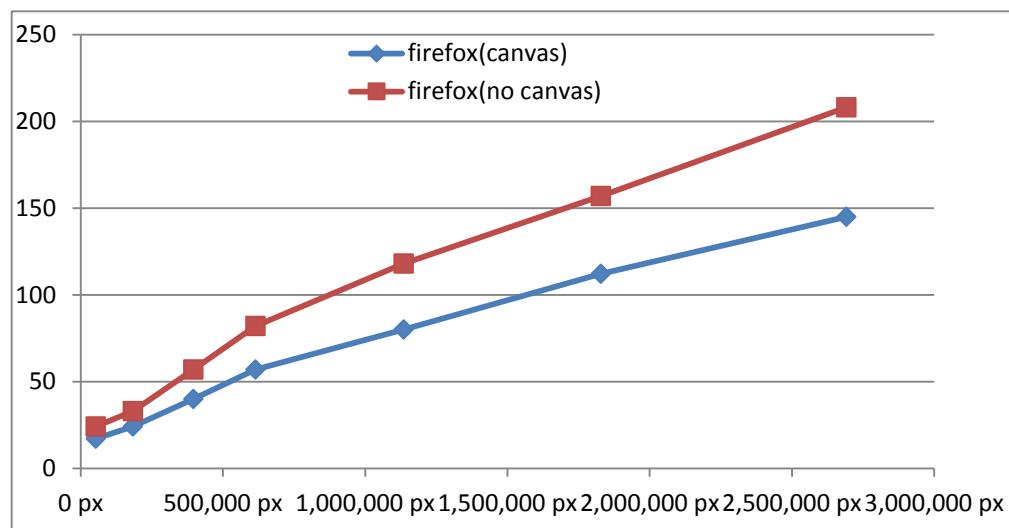
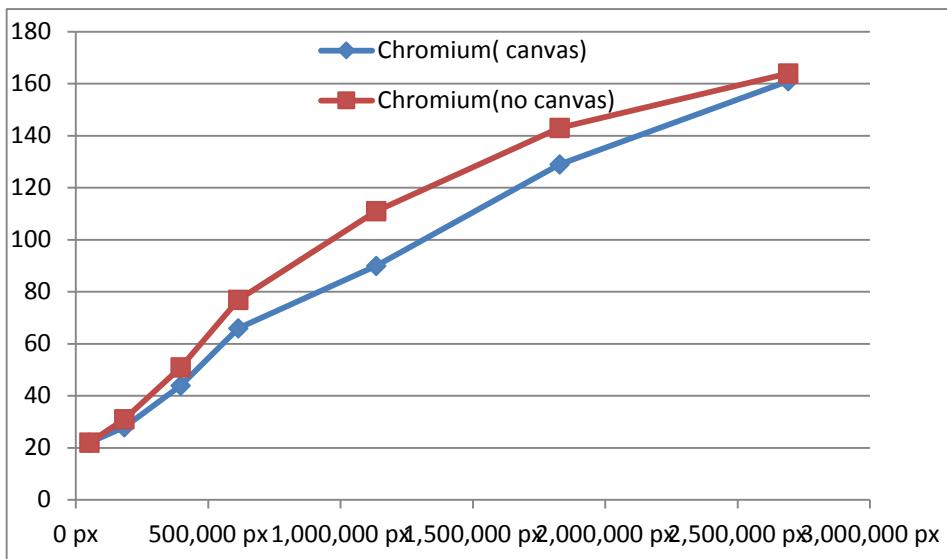


Ilustración 85 Tiempos de ejecución del procesado de las cuadrículas para el Firefox 12.

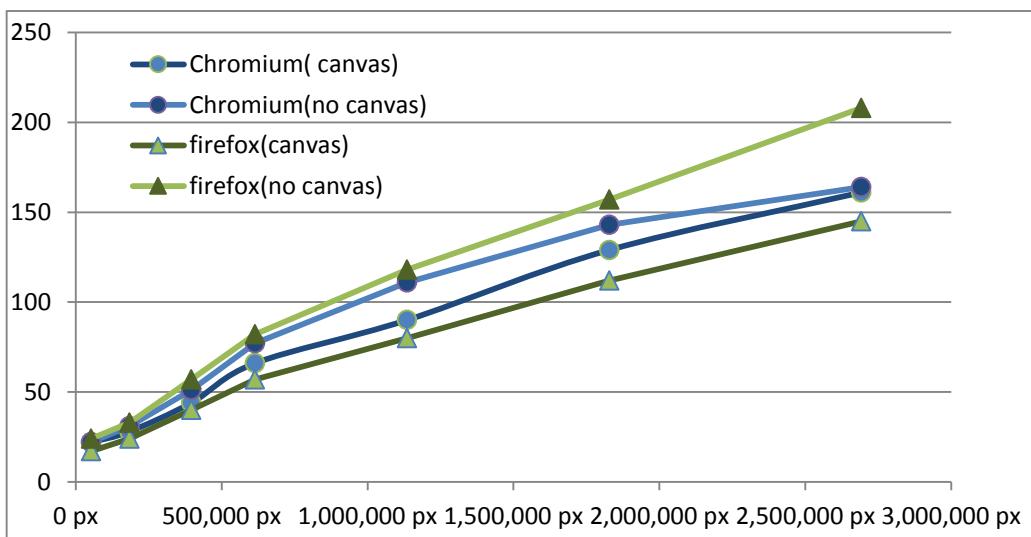
Se puede notar en la figura 7 como el enfoque de generación de cuadrículas a través del elemento Canvas es aproximadamente 25% mas rápido que mediante imágenes para el navegador Mozilla Firefox 12.

Para Chromium 18 (figura 8) los tiempos de procesado son en general inferiores que para el navegador Mozilla Firefox 12, aunque no significativos porque la tendencia de crecimiento es muy similar.



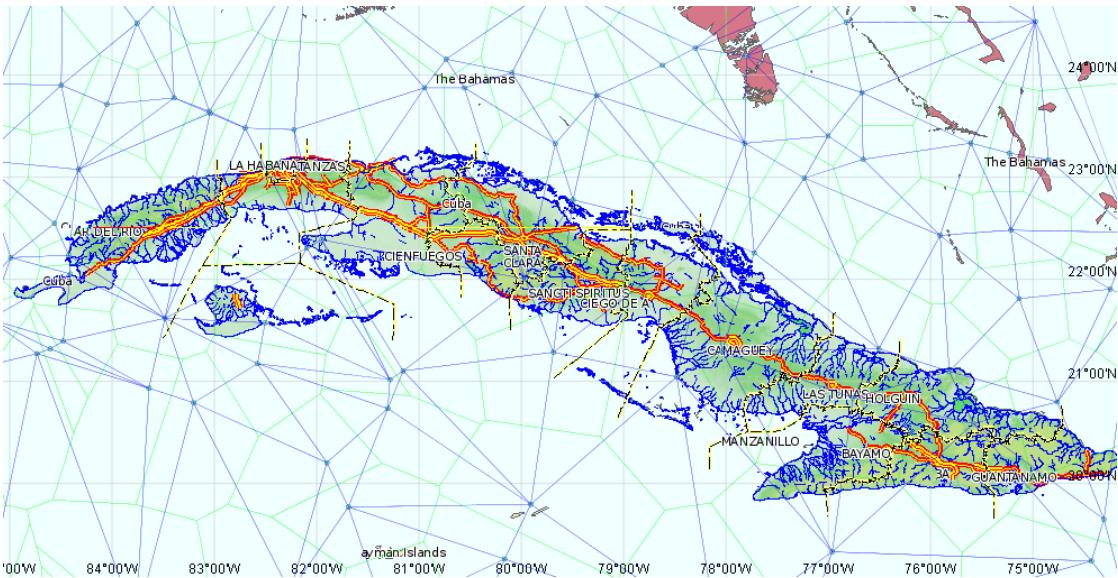
**Ilustración 86** Tiempos de ejecución procesando las cuadrículas para el navegador Chromium 18.

Para este caso, se observa un comportamiento similar para ambos enfoques de generación de cuadrículas en capas Grid de OpenLayers, aunque el enfoque Canvas es ligeramente menor. Así podemos contrastar ambos navegadores en la figura 9.



**Ilustración 87** Tiempos de ejecución procesando las cuadrículas para ambos navegadores

Como conclusión el rendimiento del procesado mediante Canvas, aunque es mayor no es muy significativo, pero nos permite otras operaciones sobre las capas que de otra forma serían casi imposible, puesto que cuando se utiliza el Canvas, los pixeles de la imagen pueden ser accedidos mediante la función `getImageData`, y modificados con `putImageData`, lo cual puede servirnos para aplicar operadores matemáticos sobre la imagen. Entre las funcionalidades que se pueden explotar, está el exportar el mapa a Imagen o PDF (figura 10) desde el mismo cliente sin necesidad de hacer peticiones al navegador.



**Ilustración 88** Imagen de un mapa exportado

#### 6.9.4 Análisis Raster en OpenLayers

Unos de los puntos débiles de OpenLayers es que largas operaciones, tienen a afectar el rendimiento de las aplicaciones, quizás por esto casi no existen tratamientos de capas Raster sobre OpenLayers. OpenLayers tiene alto acoplamiento al documento de la página, pero hay operaciones que pueden ser ejecutadas asincrónicamente en segundo plano.

##### 6.9.4.1 Operaciones sobre píxeles

Como se apreció anteriormente el nuevo enfoque de generación de cuadriculas para capas de tipo Grid, con la adición del elemento Canvas como generador de las imágenes del servicio de mapas, abre un camino a la manipulación de píxeles y procesado de estos. A modo de evitar cuelgues en las aplicaciones a medida que el número de píxeles a procesar aumente se hace uso de los WW . Como se vio anteriormente los valores de los píxeles de una imagen se almacenan en la propiedad data del objeto ImageData como un arreglo de una dimensión. Este arreglo puede dividirse uniformemente y sus valores pueden ser procesados en segundo plano fuera de la ejecución de la interfaz principal. Así mediante el método del objeto RendererContext2D getImageData se divide la imagen en filas y se distribuye el procesamiento de los píxeles en WW independientes.

Para esto se crea una clase CanvasWorker que se responsabiliza de dividir la imagen en un número determinado de filas de acuerdo a la cantidad de WW a utilizar, inicializa los WW y sincroniza el resultado de los mismos actualizando asincrónicamente la imagen original.

```

//...
for (var i = 0; i < this.countWorkers; i++) {
 var x = 0;
 var y = this.getY(i, canvasHeight);
 var height = this.getY(i+1, canvasHeight) - y;

 //...
 var imageData = this.canvasContext.getImageData(x, y, canvasWidth, height);

 if(!Renderer.workers[i]){
 var worker = new Worker(this.workerScript);
 worker.onmessage = this.onMessage(i, this.canvasContext, x, y);
 Renderer.workers[i] = worker;
 }
 var work = {
 args: this.args,
 x:x,
 y:y,
 width:canvasWidth,
 height:height,
 imageData: imageData
 }
 work.imageData.data = new Uint8Array(work.imageData.data);
 Renderer.workers[i].postMessage(work);
}

```

Observese como se ilustran los elementos mencionados en el fragmento de código anterior, donde se muestra como se divide la imagen para distribuir su procesamiento en hilos de ejecución diferentes.

#### 6.9.4.2 Análisis de generación de capas Raster

En los GIS, una de las aplicaciones es la generación de capas Raster a partir de datos vectoriales, dígase un conjunto finito de puntos en el espacio. Una de las aplicaciones se evidencia en la modelación espacial de variables continuas como la temperatura, la presión, la altura y demás. Generalmente estos procedimientos son costosos y el uso de estos sobre la ejecución del navegador podría desestabilizar e interrumpir la aplicación. Para esto, una forma diferente de generación de capas Raster se hace mediante el uso del elemento Canvas y de los WW solventando en gran medida estos problemas de inestabilidad y rendimiento.

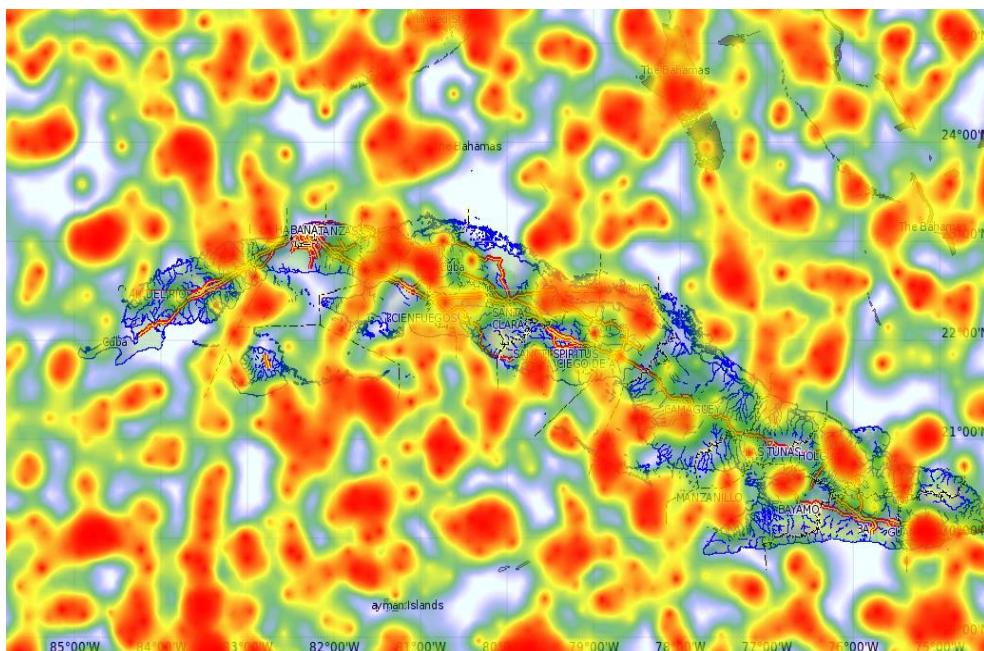
Como ejemplo tenemos la generación de una capa de densidad de puntos, donde primeramente se dibujan máscaras radiales sobre los puntos en escalas de grises con variaciones en la transparencia de acuerdo al radio de intensidad en los mismos. Después se genera un gradiente lineal de colores, mediante el cual se colorea la capa (figura 11), indexando proporcionalmente el valor de alfa en cada pixel de la imagen al color correspondiente del gradiente, según se ilustra en fragmento de código que aparece a continuación, donde se muestra la coloración y generación de la capa de densidad espacial de puntos.

```

var gradient = options.parameters.gradient;
var pix = options.imageData.data;
for (var p = 0; p < pix.length;) {
 var a = pix[p + 3] * 4;
 pix[p++] = gradient[a++];
 pix[p++] = gradient[a++];
 pix[p++] = gradient[a++];
 pix[p++] = gradient[a++];
}
options.imageData.data = new Uint8Array(pix);

```

Se elabora un caso de prueba para medir el rendimiento en la generación de la capa de densidad de puntos siguiendo un enfoque de ejecución secuencial y un enfoque de ejecución concurrente a través de los WW. Para esto se usan los navegadores Firefox 12 y Chromium 18, en una PC con sistema Operativo Ubuntu12.04, 2 GB memoria RAM, procesador Intel Dual Core 2.70 GHz, y el resultado esperado es generar una imagen de 1040x560 pixeles.

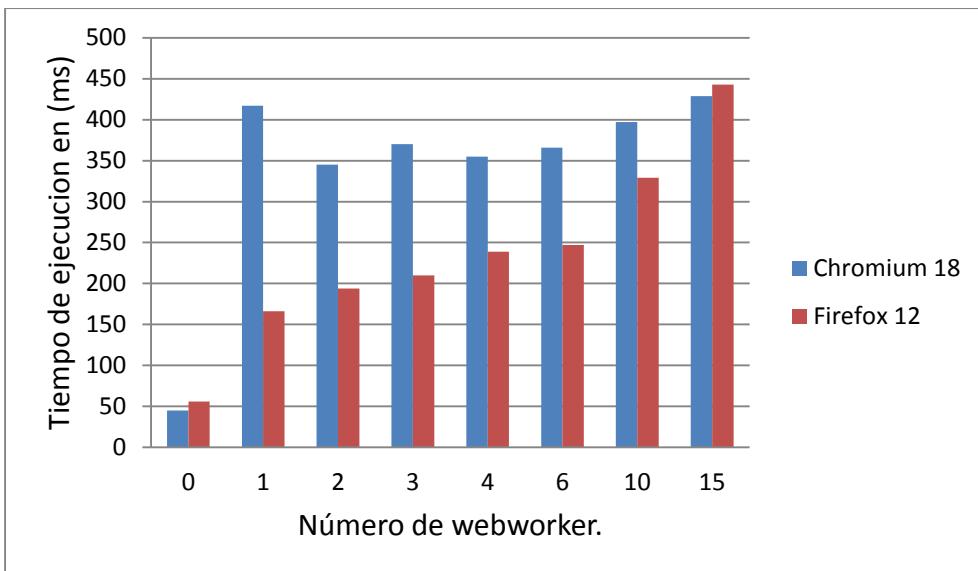


**Ilustración 89** Capa de densidad de puntos generada mediante el elemento Canvas y WW

Según se muestra en la figura.12 se aprecia que la velocidad de ejecución del enfoque secuencial es más rápido, Esto explica que para este caso, donde la complejidad algorítmica en la generación de la capa es linealmente proporcional al número de pixeles de la imagen y la cantidad de pixeles a procesar es menor de 1 millón, es factible usar el modo secuencial. Así para un procesamiento de datos mayor, una imagen dos veces mas grande que esta (2080\*1120 superando el millón de píxeles) el script se interrumpe por lo cual es preciso usar un enfoque distribuido preferiblemente en el caso de Chromium de cuatro WW y en caso de Firefox utilizar sólo un WW .

Obsérvese además que cuando el número de WW aumenta, también aumenta el tiempo de generación de la capa, esto se debe a la demora en la transferencia de datos de los WW al script principal. Es preciso destacar que no se usa la propiedad transferable<sup>39</sup> que mejora la clonación de objetos, porque no es estándar para ambos navegadores.

<sup>39</sup> <http://dev.w3.org/html5/spec/common-dom-interfaces.html#transferable-objects>



**Ilustración 90** Tiempo de ejecución de los navegadores Firefox 12 y Chromium 18 en el cálculo y procesado de la capa de densidad de puntos.

En el caso de Chromium el tiempo empleado es mayor para los primeros WW, cuando el número de WW tiende a ser muy elevado (mayor a 10) el tiempo de ejecución es similar, por tanto no es factible usar un número elevado de WW.

## 6.10 Conclusiones

Si bien es cierto que el elemento Canvas de HTML5 da soporte a nuevas formas de generación de capas, es importante señalar que es un error sustituir a la generación mediante SVG. La generación de capas depende del tipo de variable a representar, así para variables de naturaleza continua, como mapas de relieve, temperatura, presión, ... la representación mediante Canvas es la mejor opción, mientras que para representar valores discretos de una variable como gasolineras existentes en una determinada región, el mejor representador es SVG.

Es importante señalar además que la mayor parte de los navegadores modernos traen aceleración gráfica por hardware incluida en los temas de representación mediante el elemento Canvas, y es una importante característica que podría utilizarse.

Los WW y el Canvas merecen especial atención cuando se combinan, así mediante los WW pueden ejecutarse operaciones sobre los pixeles en segundo plano sin interrumpir la ejecución de la aplicación. Gracias al uso de Canvas pueden ejecutarse muchas operaciones en el contexto del navegador como exportar el mapa como imagen, realizar operaciones sobre capas como interpolación IDW<sup>40</sup>, generar mapas de densidad entre otras operaciones.

El almacenamiento local LocalStorage<sup>41</sup>, puede utilizarse para generar cache temporal de mapas y optimizar el número de peticiones al servidor de mapas.

## 7. Sistema para la generación de caché de mapas

<sup>40</sup> [http://en.wikipedia.org/wiki/Inverse\\_Distance\\_Weighting](http://en.wikipedia.org/wiki/Inverse_Distance_Weighting)

<sup>41</sup> <https://developer.mozilla.org/en/DOM/Storage>

Para poner en funcionamiento el servicio de teselas empleando la herramienta Tilecache se decidió utilizar el servidor web Apache, mediante la correcta configuración del módulo desarrollado para que este servidor HTTP sea capaz de interpretar código escrito en lenguaje de programación python (llamado mod-python), lenguaje en el que está desarrollada la aplicación.

Este documento pretende ser una guía práctica para llevar a cabo la instalación y configuración del sistema basado en teselas, de forma que este sea capaz de generar caché de mapas cartográficos para Mapserver y para los sistemas basados en teselas. Además se pretende explicar cómo se debe realizar la configuración del fichero de configuración *mapfile* para que utilice la caché de mapas como fuente de datos. Todo el proceso de instalación se realizará sobre sistema operativo Ubuntu 9.10.

## 7.1. Instalación del módulo de Apache2.0 para Python

Para llevar a cabo la instalación y configuración del módulo de Apache para *python* es necesario tener privilegios de administración; estando en el sistema operativo como administrador solo queda ejecutar los pasos para la instalación:

Se ejecuta un terminal yendo al menú **Aplicaciones -> Accesorios -> Terminal**

Para instalar Apache con el módulo de python se ejecuta como administrador el comando apt-get:

```
$ sudo apt-get install apache2 libapache2-mod-python
```

Una vez instalado todo el software, únicamente nos quedará definir en la configuración del host virtual, la configuración de TileCache , se edita el archivo:

```
$ sudo nano /etc/apache2/sites-available/default
```

Es necesario agregar las siguientes líneas:

```
<Directory /var/www/>
 Options Indexes FollowSymLinks MultiViews
 AllowOverride None
 Order allow,deny
 allow from all
 AddHandler mod_python .py
 PythonHandler mod_python.publisher
 PythonDebug On
 # This directive allows us to have apache2's default start page
 # in /apache2-default/, but still have / go to the right place
 # RedirectMatch ^/$ /apache2-default/
</Directory>
```

Para que se actualicen los cambios realizados es necesario reiniciar el servidor Apache para lo que se ejecuta:

```
$ sudo /etc/init.d/apache2 restart
```

Para comprobar que se ha instalado correctamente el módulo se crea un fichero de prueba en escrito en *python* y se ubica en el directorio /var/www llamado con el siguiente contenido

**Ejemplo:**

```
$ nano /var/www/test.py
def index(req):
 return "Test OK";
```

Se accede al fichero a través de un navegador, escribiendo la dirección

<http://localhost/test.py>, si se visualiza la cadena de caracteres 'Test OK' en la pantalla del navegador significa que se ha instalado correctamente el modulo de python para Apache.

## 7.2. Instalación de Tilecache v1.9

Tilecache es una técnica que consiste en almacenar los mapas cartográficos en forma de pequeñas imágenes que se organizan en dependencia de la escala. El almacenamiento de los mapas se conoce como caché de mapas y permite que el servidor no necesite generar una imagen cada vez que se realiza una petición. **Tilecache v1.9** es una pequeña aplicación escrita en python que funciona sobre apache. Para instalar esta aplicación es necesario seguir un grupo de pasos que se detallan a continuación:

1. Descargar la aplicación desde: <http://www.tilecache.org/>
2. El archivo comprimido que se obtiene, debe descomprimirse en la carpeta accesible por apache, típicamente en *Ubuntu* es /var/www con lo que el directorio de tilecache será: **/var/www/tilecache**
3. Al descomprimir en la carpeta tilecache hay varios archivos en el directorio. Como se está realizando la instalación con *mod\_python*, el archivo **tilecache.cgi** debe ser renombrado a **tilecache.py**.
4. Se edita el archivo /etc/apache2/sites-available/default y se añaden las siguientes líneas:

```
<Directory /var/www/tilecache/>
 AddHandler mod_python .py
 PythonHandler TileCache.Service
 PythonOption TileCacheConfig /var/www/tilecache/tilecache.cfg
 PythonPath "[var/www/tilecache/] + sys.path
 PythonDebug
</Directory>
```

On

5. Tras haber modificado este archivo se reinician los servicios de Apache:

```
$ sudo /etc/init.d/apache2 restart
```

Se puede comprobar si Tilecache está funcionando tecleando la siguiente dirección en el navegador web:

```
http://localhost/tilecache/tilecache.py?LAYERS=basic&SERVICE=WMS%20&VERSION=1.1.1&REQUEST=GetMap&SRSGEPSG:4326&BBOX=-180,-90,0,90&WIDTH=256&HEIGHT=256
```

## 7.3. Configuración del servidor de teselas

Tilecache se configura mediante un fichero de configuración llamado **tilecache.cfg**, ubicado en la raíz del paquete de clases de la propia aplicación. Este fichero emplea una serie de parámetros que son comunes para todas las capas, estos parámetros son:

- **bbox**: Caja de coordenadas rectangulares que definen el área geográfica que representa la capa correspondiente.
- **debug**: Para enviar la salida de depuración al fichero error.log. Por defecto tiene valor 'yes'.
- **description**: Descripción de la capa utilizada en algunas respuestas sobre los metadatos del servicio, por defecto es vacío.
- **extension**: Extensión o formato de las imágenes que se generarán para la capa.
- **layers**: Una cadena conformada por los nombres de las capas publicadas en el

servicio WMS para el caso de que el tipo de capa tilecache sea WMSLayer, si el tipo de capa es MapServerLayer, las capas que se especifique en este parámetro permitirán a Tilecache seleccionar que capas del fichero de mapa son las que se almacenaran en caché.

- **levels:** Un valor numérico (número entero) que representa la cantidad de niveles de acercamiento, del mapa correspondiente a la capa Tilecache en cuestión. Si se especifica el parámetro *resolutions*, este será sobreescrito.
- **mapfile:** Dirección absoluta hacia el fichero de mapa. Este parámetro se especifica para capas del tipo MapServer y Mapnik.
- **maxResolution:** Resolución máxima para la capa. Si este parámetro se especifica, el arreglo de resoluciones se calcula automáticamente a partir del valor establecido en el parámetro *levels*, teniendo en cuenta el valor establecido para la máxima resolución.
- **metaTile:** Para activar o desactivar la opción metaTiling, la cual consiste en solicitar una imagen de tamaño fuera de lo común (256x256) para conformar las teselas empleando PIL<sup>42</sup>. Soporta los valores 'yes','no' (o 'true','false').
- **metaBuffer:** Un número entero que especifica la cantidad de píxeles a solicitar fuera de los bordes de las teselas, de forma que se pueda combatir el efecto borde. Por defecto tiene como valor 10.
- **metaSize:** Valores especificados separados por una coma (2,1) que determinan el tamaño de las teselas cuando se especifica el parámetro **metaTili**. Los valores por defecto son 5,5.
- **resolutions:** Lista de resoluciones que empleará Tilecache para generar las teselas a las escalas correspondientes.
- **size:** Valores enteros separados por una coma que determinan el tamaño de las teselas. Los valores por defecto que toma este parámetro son 256,256
- **srs:** Una cadena de caracteres que especifica el sistema de referencia espacial a emplear. El valor por defecto es "EPSG:4326".
- **type:** Este parámetro especifica el tipo de capa, que puede ser: WMSLayer, MapnikLayer, MapServerLayer, imageLayer.
- **url:** Dirección URL empleada para realizar las solicitudes al servicio WMS cuando el tipo de capa es WMSLayer.
- **watermarkImage:** Dirección hacia una imagen que se aplicara sobre todas las teselas generadas. Se recomienda que la imagen que se utilice tenga el mismo tamaño que las teselas.
- **watermarkOpacity:** Opacidad para la imagen establecida en el parámetro *watermarkImage*. El valor debe estar entre 0 y 1.
- **extent\_type:** Si se especifica el valor 'loose' para este parámetro Tilecache podrá generar teselas fuera de la caja de coordenadas establecida. Se emplea para clientes que no conocen el límite para dejar de solicitar teselas
- **tms\_type:** Si se especifica el valor 'google' tilecache empleará el estilo Google x/y para generar las teselas.

### 7.3.1 Fichero de configuración tilecache.cfg

**Ejemplo:** Capa 'cache'

<pre>[cache] type base = /media/D/tilecache_local</pre>	=	<b>DiskCache</b>
---------------------------------------------------------	---	------------------

<sup>42</sup> PIL acrónimo de Python Imaging Library

**Ejemplo: Capa 'basic'**

[basic]

```
type = WMSLayer
url = http://10.12.167.23/cgi-bin/mapserv?map=/var/www/cartografia/mapa-wms.map
layers = provincia, hidroareaw
extension = png
srs = EPSG:4326
extent_type = loose

levels = 10

resolutions = 0.703125, 0.250000, 0.062500, 0.034467577934265137, 0.031250,
0.030203788967132568, 0.027233788967132568, 0.017233788967132568, 0.015625,
1.000000, 0.0086168944835662842, 0.0060168944835662842, 0.0050168944835662842,
0.0043084472417831421, 0.0033084472417831421,
0.002154223620891571, 0.001373, 0.001010, 0.000977, 0.000244, 0.000110, 0.000093, 0.0000
76, 0.000061, 0.000019, 0.000017, 0.000015, 0.000001
```

**Ejemplo: Capa 'genesig\_all'**

[genesig\_all]

type = MapServerLayer

```
mapfile = /var/www/mapserver/geowebBase/geoweb
Base.map
layers = =
extension = png
srs = =
extent_type = loose
levels = 20

resolutions = 0.703125, 0.250000, 0.062500, 0.034467577934265137, 0.031250,
0.030203788967132568, 0.027233788967132568, 0.017233788967132568, 0.015625,
1.000000, 0.0086168944835662842, 0.0060168944835662842, 0.0050168944835662842,
0.0043084472417831421, 0.0033084472417831421, 0.002154223620891571, 0.001373,
0.001010, 0.000977, 0.000244, 0.000110, 0.000093, 0.000076, 0.000061, 0.000019,
0.000017, 0.000015, 0.000001
```

**Ejemplo: Capa 'integracion'**

[integracion]

type = MapServerLayer
mapfile = /var/www/GIS/vinculando-con-mapserver/geowebBase/geowebBase.map
extension = png

layers = cubageneralr, relieve, zonasbajasr\_500, municipios\_500, hidrolinealpl\_500,
hidroarealr\_500, vialespl\_500, lineaelectricapl\_500, costalitoralpl\_500, rothidrot\_500,
rotplanit\_500, rotlitonimiat\_500, rotoronimiat\_500, rotdpat\_500, codifrotvialest\_500,
presapl\_500, pobladofp\_500, viasferreaspl\_500, simbolospuntualfp\_500, limiteterritpl\_50
0

extension=png

srs=EPSG:4267

extent\_type = loose
levels = 20

```
resolutions = 0.703125, 0.250000, 0.062500, 0.034467577934265137, 0.031250,
0.030203788967132568, 0.027233788967132568, 0.017233788967132568, 0.015625, 1.000000, 0.0086168944835662
842, 0.0060168944835662842, 0.0050168944835662842, 0.0043084472417831421, 0.0033084472417831421,
0.002154223620891571, 0.001373, 0.001010, 0.000977, 0.000244, 0.000110, 0.000093, 0.0000
76, 0.000061, 0.000019, 0.000017, 0.000015, 0.000001
```

### 7.3.2 Cálculo de resoluciones y escalas para Tilecache

Para calcular las resoluciones se emplea el algoritmo que aplica la siguiente fórmula:

$$resolucion = \frac{1}{(n * i * d)} \quad n = \frac{1}{e}$$

**Sustituyendo n por 1/e**

$$resolucion = \frac{1}{\left(\frac{1}{e} * i * d\right)}$$

$$resolution = \frac{e}{(i * d)}$$

**Donde:**

- **e:** denoma el valor escala
- **i:** hace referencia a la cantidad de pulgadas por unidad de medida
- **d:** equivale a la cantidad de puntos por pulgada (72)

**Factores de unidades de conversión 'pulgadas por unidad'**

pulgadas (inches)	1
pies (ft)	12
millas (mi)	63.360
metros (m)	39,37
kilómetros (km)	39370,1
grados decimales (dd)	4.374.754

Mediante el algoritmo que se plantea se obtuvieron las resoluciones pertenecientes a las escalas que se ilustran a continuación, pero para ello primero fue necesario realizar la selección de escalas en las que se generaría la caché, obteniendo las resoluciones de la forma:

$$resolucion = \frac{25000}{(4374754 * 72)}$$

$$resolucion = 0.00007936954220105226$$

Resoluciones obtenidas a partir de las diferentes escalas que se relacionan:

Escala	Resolución
25	0,00007936954220105220
50	0,00015873908440210500
100	0,00031747816880420900
250	0,00079369542201052300
500	0,00158739084402105000

1.000.000	0,00317478168804209000
2.000.000	0,00634956337608418000
5.000.000	0,01587390844021050000
10.000.000	0,03174781688042090000
25.000.000	0,07936954220105230000
50.000.000	0,15873908440210500000

## 7.4 Generación manual de la caché de mapas

Después que se ha instalado y configurado correctamente Tilecache, es posible emplear la utilidad *tilecache\_seed.py* para generar la caché de mapas correspondiente a las capas que se definieron en el fichero *tilecache.cfg*.

***tilecache\_seed.py <url> <layer> [<zoom\_start> <zoom\_stop> [<bbox>]]***

- **url** : Dirección de publicación del servicio de teselas

**Ejemplo:** [http://10.12.167.23/tilecache\\_local/tilecache.py?](http://10.12.167.23/tilecache_local/tilecache.py?)

- **Layer**: Nombre de la capa de la que se desea obtener las teselas, la misma debió haber sido especificada en el fichero de configuración *tilecache.cfg*.
- **zoom\_start** : Nivel de acercamiento desde el cual se desea comenzar a generar las teselas.
- **zoom\_end** : Nivel de acercamiento que define hasta que nivel de acercamiento se desea generar las teselas.
- **bbox**: La caja de coordenadas rectangulares que definen el área geográfica que se desea almacenar en caché de mapas.

A continuación se especifica cómo hacer uso de esta utilidad a través de líneas de comandos:

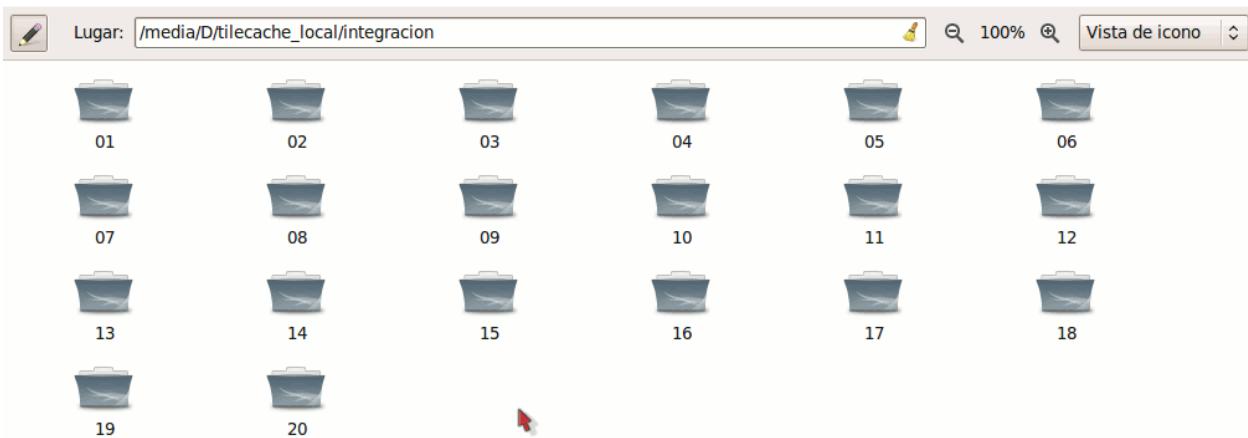
1. Ubicarse en el directorio de instalación de la aplicación *Tilecache*

```
$ cd /var/www/tilecache_local
```

2. Ejecutar la funcionalidad *tilecache\_seed.py* pasandole los parametros necesarios:

```
$ tilecache_seed.py http://10.12.167.23/tilecache_local/tilecache.py? Integracion 1 21 - 90.00,15.00,-70.00,27.00
```

Después de ejecutar la utilidad para generar la caché de mapas, dependiendo de la dirección establecida en el parámetro **base** de la capa **caché**, se generará una carpeta por cada capa definida, dentro de la cual se creará una carpeta por cada escala del mapa que se generó. De esta forma el servidor de teselas puede identificar las teselas correspondientes ante las solicitudes de los usuarios. Para el caso del ejemplo presentado en la sección **2.1.1**, la caché de mapas se genera en el directorio '*/media/D/tilecache\_local*'.

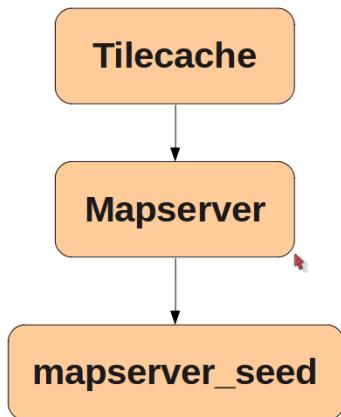


**Ilustración 91** Caché de mapa por escala

## 7.5 Generando la caché para Mapserver

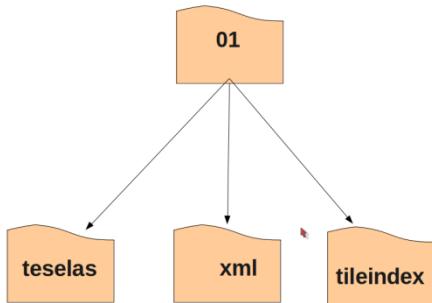
Para generar la caché para que el servidor de mapas *Mapserver* sea capaz de interpretar los datos, se necesita añadirle una clase a *Tilecache* denominada '*Mapserver*' contenida en un fichero escrito en lenguaje de programación *python* llamado '*Mapserver.py*'.

Para realizar las solicitudes al servicio *WMS-C* es necesario agregar una funcionalidad '*mapserver\_seed*' capaz de emplear la herramienta *GDAL* para traducir los ficheros *XML* generados, así como almacenar de forma organizada, por cada una de las escalas del mapa, y empleando un directorio común para la caché de mapas.



**Ilustración 92** Funcionalidad de Mapserver

Para almacenar la caché en disco se propone una estructura de carpetas donde cada paquete se organiza jerárquicamente respetando las capas y escalas en dependencia de la información espacial que se almacena. Se propone estructurar la caché de mapas para *Mapserver* en un directorio común donde existirá un paquete por cada capa especificada en el fichero de configuración de *Tilecache*, cada paquete tendrá a su vez tres paquetes que serán comunes para todas las capas cuya caché sea generada.



**Ilustración 93** Estructura de carpetas por escala

A continuación se describe la función asociada a cada uno de los paquetes generados para cada capa:

- **xml**

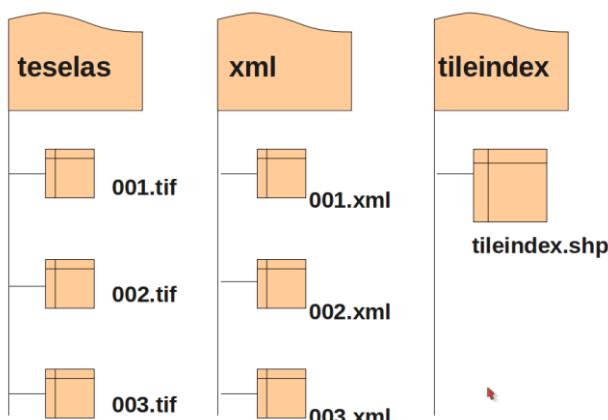
En el paquete denominado 'xml' se almacenan todos los ficheros en formato XML pertenecientes a cada una de las teselas generadas mediante la utilidad 'tilecache\_seed', cada archivo contiene la información referente al servicio de teselas publicado a través de Tilecache así como la caja de coordenadas rectangulares que define la tesela correspondiente. Estos ficheros son necesarios porque permiten realizar la traducción a otros ficheros en formato raster que representan cada una de las teselas generadas pero con información geográfica asociada para que puedan ser interpretados por el servidor de mapas.

- **teselas**

El paquete 'teselas' es el encargado de almacenar las teselas una vez que se ejecuta la funcionalidad 'mapserver\_seed' agregada junto con la clase 'Mapserver'. Esta funcionalidad tiene como función principal recorrer el directorio 'xml' e ir traduciendo los ficheros XML de forma que se consuma el servicio de teselas publicado, y se obtengan las imágenes para almacenarlas posteriormente con los datos espaciales contenidos en las propias imágenes.

- **tileindex**

En el paquete 'tileindex' se almacenan los archivos en formato shape que funcionan como puente entre el servidor de mapas y los datos almacenados en forma de teselas.



**Ilustración 94** Estructura de paquetes por escala

### 7.5.1 Funcionalidad mapserver\_seed.py

Para generar la caché de mapas para el servidor mapserver es necesario hacer uso de la versión modificada de Tilecache (v1.9.1), para obtenerla se puede contactar al autor *Eduanys Puerta Ramos* mediante el correo electrónico [epuerta@uci.uci.cu](mailto:epuerta@uci.uci.cu).

La nueva versión de Tilecache permite generar los directorios explicados en el epígrafe anterior, mediante la clase Mapserver. La funcionalidad tilecache\_seed.py fue modificada para generar los ficheros XML necesarios para llevar a cabo la traducción de las teselas generadas por Tilecache a imágenes georreferenciadas que el servidor de mapas puede utilizar para generar los mapas. Por tanto una vez que se ha ejecutado tilecache\_seed.py, debieron haber sido generados los ficheros .xml sin dificultades, solo queda ejecutar mapserver\_seed.py para traducir los ficheros y crear los directorios *teselas* y *tileindex*.

### 7.5.2 Ejecutando mapserver\_seed.py

La funcionalidad mapserver\_seed.py tiene la siguiente sintaxis, para ejecutarla a través de un terminal:

`mapserver_seed.py <cache_path> <layer> zoom_start zoom_end`

- **cache\_path**: directorio especificado para almacenar la caché de mapas para mapserver.
- **Layer**: capa Mapserver que contiene los ficheros XML a traducir
- **zoom\_start** : Nivel de acercamiento desde el cual se desea comenzar a generar las teselas.
- **zoom\_end** :Nivel de acercamiento que define hasta que nivel de acercamiento se desea generar las teselas.

A continuación se especifica cómo hacer uso de esta utilidad a través de líneas de comandos:

1. Ubicarse en el directorio de instalación de la aplicación *Tilecache*

```
$ cd /var/www/tilecache_local
```

2. Ejecutar la funcionalidad mapserver\_seed.py pasándole los parámetros necesarios:

```
$ mapserver_seed.py /media/D/mapserver-cache integracion 1 20
```

Después de ejecutar la utilidad para generar la caché de mapas, por cada paquete generado por escala se crearán los directorios:

- /media/D/mapserver-cache/integracion/escala/tiles
- /media/D/mapserver-cache/integracion/escala/tileindex

Por tanto queda disponible la caché de mapas para Mapserver, siendo esta accesible desde un fichero de mapa (mapfile), donde se establecerán los *shapefiles* generados, como fuente de datos.

## 7.6 Tileindex

Un *tileindex* es un archivo en formato '.shp' que une varios conjuntos de datos en una sola capa permitiendo de esta forma no tener que especificar una capa distinta para cada imagen que se quiera cargar en el mapa. Este método permite que el servidor de mapas una las piezas y realice un mosaico a partir de imágenes georreferenciadas.

Para que el servidor de mapas sea capaz de generar los mapas cartográficos a partir de

la caché, es necesario establecer la configuración adecuada en el fichero de mapa con extensión '.map'. Para establecer dicha configuración *Mapserver* permite que en el *mapfile* sean creadas capas cuyas fuentes de datos pueden ser los *tileindex* generados mediante la funcionalidad *mapserver\_seed*. Utilizando un *tileindex* en el fichero de configuración:

```
LAYER
 NAME "Caminos"
 STATUS ON
 TYPE LINE
 TILEINDEX "tileindex.shp"
 TILEITEM "LOCATION"
END
```

Para mayor información sobre los *tileindex* se debe visitar el sitio oficial del servidor de mapas Mapsever: <http://mapserver.org/optimization/tileindex.html>.

### 7.6.1 Configuración del fichero de \*.map

Con los ficheros en formato shape generados por cada una de las escalas, solo queda configurar un fichero de mapa para que utilice dichos shapefiles como fuente de datos para generar los mapas. Como cada fichero '.shp' hace referencia a las teselas correspondientes a una escala determinada se debe establecer la denominación adecuada para cada capa de tipo *tileindex*, es decir se debe establecer el rango de escala en el que se visualiza la capa.

A continuación se muestran algunas capas de tipo *tileindex* configuradas para utilizar la caché de mapas como fuente de datos:

**Ejemplo:** Capa a nivel 06, escalas 9000000- 9300000

```
LAYER
 MINSCALEDENOM 9000000
 MAXSCALEDENOM 9300000
 NAME "tileindex06"
 GROUP "Tileindex"
 STATUS on
 TILEINDEX "/media/D/mapserver-cache/integracion/06/tileindex/06.shp"
 TILEITEM "location"
 TYPE RASTER
 PROCESSING "RESAMPLE=AVERAGE"
 OFFSITE 255 255 255
 METADATA
 "wms_title" "drg"
 END
 PROJECTION
 "init=epsg:4267"
 END
END
```

## 7.7 Conclusiones

Después de haber puesto en funcionamiento los sistemas propuestos para generar caché de mapas cartográficos se puede concluir que estableciendo correctamente los parámetros de configuración tras haber instalado la aplicación modificada Tilecache

v1.9.1, es posible generar caché de mapas cartográficos con teselas georreferenciadas, de forma que los visores de mapas y aplicaciones encargadas de renderizar imágenes representativas de mapas cartográficos, pueden hacer uso de dicha caché sin dificultades.

## 8 Manipulación de datos espaciales

Es un sistema administrador de bases de datos que maneja datos de tipo geoespacial, entendiéndose por éste, como un objeto que en el espacio establece un marco de referencia para definir su localización y su relación con otros objetos. El espacio comúnmente usado es el espacio físico que es un lugar manipulable, perceptible y que sirve de referencia para el análisis geoespacial.

La construcción de una base de datos geográfica implica un proceso de abstracción, para pasar del mundo real a una representación matemática y más simple que pueda ser procesada por algún lenguaje de computadora diseñado para este fin. Este proceso de abstracción tiene diversos niveles y normalmente comienza con la concepción de la estructura de la base de datos, generalmente en capas, las cuales se clasifican según su información temática, para su posterior inclusión en algún análisis de su información.

La estructuración de la información espacial procedente del mundo real en capas conlleva cierto nivel de dificultad. En primer lugar, la necesidad de abstracción que requieren los computadores implica trabajar con geometrías básicas de dibujo, de tal forma que toda la complejidad de la realidad ha de ser reducida a puntos, líneas o polígonos. En segundo lugar, existen relaciones espaciales entre los objetos geográficos que el sistema no puede obviar, la topología, que en realidad es el método matemático-lógico usado para definir las relaciones espaciales entre los objetos geográficos.

Los atributos gráficos son guardados en archivos y manejados por el software de un sistema SIG. Los objetos geográficos son organizados por temas de información, o capas de información, llamadas también niveles. Los puntos, líneas y polígonos son almacenados en capas separadas, permitiendo la agrupación de la información en temas junto con los atributos no gráficos, es decir, información que esta asociada pero que no representa un dato espacial. Un ejemplo de este tipo de clasificación puede ser predios y lagos pertenecientes a objetos de “Polígonos”, o ríos y carreteras pertenecientes a objetos “líneas”.

En la actualidad los formatos estándar para el almacenamiento de la información espacial son los formatos RASTER y el formato VECTOR. En el primero de ellos, se define una grilla o una malla de rectángulos o cuadrados a los que se les llama píxeles, cada píxel tiene asociado información digital (principalmente color) que agrupados representan las características de la zona o superficie geográfica que cubre, si queremos asociarlo a la matemática es comparable con una matriz de números y cada valor numérico tiene su equivalencia en el aspecto geográfico. Como ejemplos de este formato se pueden citar la salida de un proceso de imagen satelital y una fotografía aérea.

El formato vectorial representa la información por medio de pares ordenados de coordenadas, este ordenamiento da lugar a figuras geométricas con las que se representan los objetos gráficos, así, un punto se representa mediante un par de coordenadas, una línea con dos pares de coordenadas, una polilínea como una serie de líneas y un polígono como una polilínea cerrada. A estas geometrías, se les puede asociar atributos y almacenar éstos en una base de datos destinada para tales propósitos.

**Métodos de acceso espacial:** Para evitar la búsqueda repetitiva de la información en una base de datos, se crean índices, que reducen el número de elementos a revisar en la base de datos en un procesamiento de consulta, logrando con esto mayor eficiencia en los tiempos de respuesta. Es por este motivo que existen tres categorías de métodos de acceso espacial, las PAM<sup>43</sup>, R-Tree, y SAM<sup>44</sup>, los cuales se utilizan de acuerdo al tipo de dato en el que está la base de datos espacial ya sea raster o vectorial.

**Aplicaciones:** La información geográfica contiene una referencia territorial explícita como por ejemplo la latitud y longitud, o una referencia implícita como domicilio o código postal. Las referencias implícitas pueden ser derivadas de referencias explícitas mediante geocodificación. La información geográfica es a su vez el elemento diferenciador de un Sistema de Información Geográfica frente a otro tipo de Sistemas de Información, así la particular naturaleza de este tipo de información contiene dos caminos diferentes, por un lado está el camino espacial y por otro el camino temático de los datos. Mientras otros Sistemas de Información contienen sólo datos alfanuméricos (nombres, direcciones, números de cuenta, etc.), las bases de datos de un SIG integran además la delimitación espacial de cada uno de los objetos geográficos.

A continuación se describen dos diferentes aplicaciones en el uso de bases de datos espaciales:

**Los denominados SIG Puros:** Son bases de datos espaciales sin ninguna capa intermedia, realizan las operaciones de selección espacial de manera nativa. Son modulares, extensibles y normalmente con una interfaz amigable. Aunque también son capaces de generar una interfaz gráfica amigable para las bases de datos comunes, de tal manera de utilizar datos espaciales ya almacenados en estas tecnologías.

**Bases de datos con extensiones para bases de datos espaciales:** Son sistemas de bases de datos normales a los cuales se les agrega una capa para el manejo de la geometría y realizar el "traspaso" desde datos comunes a datos espaciales transparente al usuario.

**PostGIS** es una extensión al sistema de base de datos objeto-relacional PostgreSQL. Permite el uso de objetos SIG. PostGIS incluye soporte para índices GiST basados en R-Tree, y funciones básicas para el análisis de objetos SIG. Esta creado por Refractions Research Inc, como un proyecto de investigación de tecnologías de bases de datos espaciales. Está publicado bajo licencia GNU. Con PostGIS podemos usar todos los objetos que aparecen en la especificación OGC 7 como puntos, líneas, polígonos, multilíneas, multipuntos, y colecciones geométricas.

Esta herramienta se monta sobre el servidor de datos PostgreSQL y nos permite manejar nuevos tipos de datos que no son parte del motor. Existen versiones de PostgreSQL para Linux y Windows, al igual existen versiones de PostGIS para ambos sistemas operativos.

## 8.1 Objetos GIS

Los objetos GIS soportados por PostGIS son de características simples definidas por OpenGIS. Actualmente PostGIS soporta las características y el API de representación de

<sup>43</sup> PAM acrónimo de Point Access Method

<sup>44</sup> SAM acrónimo de Spatial Access Method

la especificación OpenGIS pero no tiene varios de los operadores de comparación y convolución de esta especificación.

Ejemplos de la representación en modo texto:

```
POINT(0 0 0)
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
MULTIPOINT(0 0 0,1 2 1)
MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0), (1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)), ((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
GEOMETRYCOLLECTION(POINT(2 3 9), LINESTRING((2 3 4,3 4 5))
```

En los ejemplos se pueden ver características con coordenadas de 2D y 3D, ambas son permitidas por PostGIS. Tambien posible emplear las funciones *force\_2d()* y *force\_3d()* para convertir una característica a 3d o 2d.

### 8.1.1 Forma CANONICA vs ESTANDAR.

OpenGIS define dos formas de representar los objetos espaciales: WKT<sup>45</sup> como en ejemplos anteriores y WKB<sup>46</sup>. Las dos formas guardan información del tipo de objeto y sus coordenadas. Además la especificación OpenGIS requiere que los objetos incluyan el identificador del sistema de referencia espacial (SRID). El SRID es requerido cuando insertamos un objeto espacial en la base de datos.

```
INSERT INTO SPATIALDATABASE(THE_GEOM, THE_NAME)
VALUES(GeometryFromText('POINT(-126.4 45.32)', 312), 'Un Lugar')
```

La función *GeometryFromText* requiere un numero SRID. En PostgreSQL tenemos la representación en *forma canónica*, es una representación en modo texto. Esta representación es distinta al estándar openGIS. A continuación se muestran algunas diferencias.

```
db=> SELECT AsText(geom) AS OGCGeom FROM thetable;
OGCGeom

LINESTRING(-123.741378393049 48.9124018962261,-123.741587115639 48.9123981907507)
(1 row)

db=> SELECT geom AS PostGISGeom FROM thetable;
PostGISGeom

SRID=123;LINESTRING(-123.741378393049 48.9124018962261,-123.741587115639 48.9123981907507)
(1 row)
```

### 8.2 Usar el estándar OpenGIS.

Las especificación para SQL de características simples de OpenGIS define tipos de objetos GIS estándar, los cuales son manipulados por funciones, y un conjunto de tablas de meta- datos. Hay 2 tablas de meta-datos en la especificación OpenGIS:

<sup>45</sup> WKT acrónimo de Well Know Text

<sup>46</sup> WKB acrónimo de Well Know Binary

## 8.2.1 SPATIAL\_REF\_SYS

Esta tabla contiene un identificador numérico y una descripción textual de el sistema de coordenadas espacial de la base de datos, tal y como se define a continuación:

```
CREATE TABLE SPATIAL_REF_SYS(
 SRID INTEGER NOT NULL PRIMARY KEY,
 AUTH_NAME VARCHAR(256),
 AUTH_SRID INTEGER,
 SRTEXT VARCHAR(2048),
 PROJ4TEXT VARCHAR(2048)
)
```

A continuación se describen los elementos que la componen:

- **SRID:** Valor entero que identifica el sistema de referencia espacial.
- **AUTH\_NAME:** El nombre del estándar para el sistema de referencia. Por ejemplo: EPSG.
- **AUTH\_SRID:** El identificador según el estándar AUTH\_NAME. En el ejemplo anterior es el id según EPSG.
- **PROJ4TEXT:** Proj4 es una librería que usa PostGIS para transformar coordenadas. Esta columna contiene una cadena con definición de las coordenadas de Proj4 para un SRID dado.

+proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m

- **SRTEXT:** Define una representación basada en la especificación *Well Known Text* para el sistema de referencia espacial.

Ejemplo: WKT para SRS.

```
PROJCS[
 "NAD83/UTM Zone 10N",
 GEOGCS[
 "NAD83",
 DATUM[
 "North_American_Datum_1983",
 SPHEROID["GRS 1980", 6378137,298.257222101]
],
 PRIMEM["Greenwich", 0],
 UNIT["degree", 0.0174532925199433]
],
 PROJECTION["Transverse_Mercator"],
 PARAMETER["latitude_of_origin",0],
 PARAMETER["central_meridian",-123],
 PARAMETER["scale_factor",0.9996],
 PARAMETER["false_easting",500000],
 PARAMETER["false_northing",0],
 UNIT["metre",1]
```

]

Para obtener una lista mas detallada de las representaciones WKT de EPSG, se puede consultar el sitio: <http://www.opengis.org/techno/interop/EPSG2WKT.TXT>.

## 8.2.2 GEOMETRY\_COLUMNS

El término de geometry\_columns se define como tabla geométrica o contenedor de datos espaciales las cuales deben contener una serie de campos bien definidos, en función de una correcta interpretación de sus datos, a continuación se muestra su sintaxis.

```
CREATE TABLE GEOMETRY_COLUMNS(
 F_TABLE_CATALOG VARCHAR(256) NOT NULL,
 F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
 F_TABLE_NAME VARCHAR(256) NOT NULL,
 F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
 COORD_DIMENSION INTEGER NOT NULL,
 SRID INTEGER NOT NULL,
 TYPE VARCHAR(30) NOT NULL
)
```

A continuación se procede a describir cada uno de los elementos que conforman una tabla geométrica:

- **F\_TABLE\_CATALOG, F\_TABLE\_SCHEMA, F\_TABLE\_NAME:** Distingue totalmente la tabla de características que contiene la columna geométrica.
- **F\_GEOMETRY\_COLUMN:** Nombre de la columna geométrica en la tabla de características.
- **COORD\_DIMENSION:** Dimensión espacial de la columna (2D o 3D).
- **SRID:** Es una clave foránea que referencia SPATIAL\_REF\_SYS.
- **TYPE:** Tipo de objeto espacial, los cuales pueden tomar valores comprendidos en el siguiente rango [POINT / LINESTRING / POLYGON / MULTIPOLY / GEOMETRYCOLLECTION], en caso de ser requerido un tipo heterogéneo se recomienda emplear el GEOMETRY.

## 8.2.3 Crear una tabla espacial

Una de las vías tradicionales definida para la construcción de tablas de datos espaciales consiste en dos pasos principales. Lo primero que debe hacerse es crear una tabla a través del comando *CREATE TABLE*, luego se procede a adicionar el campo geométrico a través de la función *AddGeometryColumn* definida por la OpenGIS. La cual debe recibir por parámetro el nombre de la base de datos, la tabla y la columna deseada, seguido debe especificarse el srid, el tipo, así como la dimensión. Para mayor comprensión véase el ejemplo que se muestra a continuación:

```
CREATE TABLE CALLES_GEO(ID int4, NAME varchar(25))
SELECT AddGeometryColumn('calles_db', 'calles_tb_geom', 'cl_geom', 423, 'LINESTRING', 2)
```

En caso de requerir un campo geométrico de carácter genérico sería de la siguiente forma:

```
CREATE TABLE CALLES(CALLE_ID int4, CALLE_NOMBRE varchar(128));
SELECT AddGeometryColumn('calles_db','calles','calles_geom',-1,'GEOMETRY',3);
```

## 8.3 Importar datos datos espaciales

En sentido general los datos espaciales pueden ser concentrados en una amplia gama de formatos. En función de esto la comunidad internacional ha puesto sus esfuerzos y se han logrado desarrollar algunas herramientas que facilitan estos procesos, sin embargo en la actualidad no existe el método definitivo para cubrir esta gran diversidad a la que

podemos estar enfrentándonos.

El método más básico se refiere al empleo del lenguaje SQL, el cual está orientado principalmente para cuando se tienen los datos en texto plano, permitiendo realizar una inserción masiva de dicha información a través del recurso *INSERT* y la función *GeometryFromText*. Esta última permite obtener el valor geométrico de forma binaria a partir de la representación textual del mismo. A continuación se muestra un ejemplo que ilustra su utilización

```
BEGIN;
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(191232 243118,191108 243242)', -1), 'Jeff Rd');
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(189141 244158,189265 244817)', -1), 'Geordie Rd');
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(192783 228138,192612 229814)', -1), 'Paul St');
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(189412 252431,189631 259122)', -1), 'Graeme Ave');
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(190131 224148,190871 228134)', -1), 'Phil Tce');
 INSERT INTO CALLES_GEO(ID,GEOM,NAME) VALUES (1, GeometryFromText
 ('LINESTRING(198231 263418,198213 268322)', -1), 'Dave Cres');
END;
```

Este fragmento de código puede ser almacenado en un fichero de extensión *.sql*, y ser ejecutado desde la interfaz de líneas de comando, específicamente empleando el recurso provisto por PostgreSQL denominado *psql*. A dicho comando se le debe especificar el nombre de la base de datos definida por el parámetro *-d* y el directorio del fichero que contiene las instrucciones definido por *-f*, a continuación se muestra un ejemplo.

```
psql -d calles -f calles.sql
```

Otro de los formatos muy comunes en que se distribuye la información cartográfica es el conocido *shape*, para el cual se desarrolló la herramienta *shp2pgsql*. Esta permite transformar el contenido de los archivos de figuras *ESRI* a *SQL* para su inserción en una base de datos *PostGIS/PostgreSQL*. El cargador tiene varios modos de operación que se seleccionan con los parámetros desde el *CLI*, los cuales se describen a continuación.

- **-d:** Elimina la tabla de la base de datos antes de crear la tabla con los datos del archivo de figuras.
- **-a:** Añade los datos del archivo de figuras a las tablas de la base de datos. El fichero debe tener los mismos atributos que la tabla.
- **-c:** Crea una nueva tabla y llena esta con el archivo de figuras. Este es el modo por defecto.
- **-D:** Crea una tabla nueva llenándola con los datos del fichero de formas. Pero usa el formato dump para la salida de datos que es más rápido que el *insert* de *SQL*. Se usará este para conjuntos de datos largos.
- **-s:** Crea y rellena una tabla geométrica con el SRID que se le pasa como parámetro.

En caso de que se desee

```
shp2pgsql shapecalles tablacalles callesdb > calles.sql
psql -d callesdb -f calles.sql
```

También se puede efectuar a través de un comando múltiple definido como:

```
shp2pgsql shapecalles tablacalles callesdb | psql -d callesdb
```

## 8.4 Exportar datos espaciales

De igual forma que la importación de contenido geográfico, existen muchos métodos de exportación. El más básico consiste en emplear consultas simples en lenguaje *sql* a partir del recurso nativo denominado *SELECT*, utilizando la función *AsText* sobre el campo definido en función de almacenar el contenido geométrico, tal y como se ilustra en el ejemplo que se muestra a continuación.

```
SELECT id, AsText(geom) AS geom, name FROM ROADS_GEOM;
```

En el caso de restricciones basadas en los atributos usamos la misma sintaxis que para tablas no espaciales. Para restricciones espaciales tenemos los siguientes operadores:

- **&&:** Indica cuando la caja que contiene una geometría que se superpone a la caja de otra.
- **~=:** Define si dos geometrías son idénticas, por ejemplo de especificarse '[POLIGON\(\(0 0,1,1 0,0\)\)](#)'~='[POLIGON\(\(0 0,1,1 0,0\)\)](#)'
- **=:** Indica si las cajas circunscritas de dos geometrías son iguales

Para usar estos operadores debemos cambiar la representación del formato de texto a geometrías usando la función *GeometryFromText()*. Por ejemplo en caso de requerirse todos los registros que sean idénticos a una geometría específica, la sintaxis debería ser similar a la que se muestra a continuación:

```
SELECT ID,NAME FROM CALLES_GEO
WHERE GEOM ~= GeometryFromText('LINESTRING(191232 243118,191108 243242)',-1);
```

Cuando usamos el operador **&&**, podemos especificar como característica de comparación una *BOX3D* o una *GEOMETRY*, tal y como se muestra a continuación:

```
SELECT ID, NAME FROM CALLES_GEO
WHERE GEOM && GeometryFromText('POLYGON((1912 2417, 1932 2419,19124 2317,1912 2317))', -1);
```

La consulta más usada es la basada en marcos, que usan los clientes para mostrar un marco del mapa. Podemos usar un *BOX3D* para el marco:

```
SELECT AsText(GEOM) AS GEOM FROM ROADS_GEO
WHERE GEOM && GeometryFromText('BOX3D(191232 243117,191232 243119)::box3d, -1);
```

Observese que el valor **-1** indica que no se especifica ningún SRID.

Al igual que en el acápite anterior podemos requerir exportar nuestros datos a formato de *ESRI* o también conocido como ficheros *shape*. Para lograr esto podemos utilizar la utilidad denominada *pgsql2shp*, la cual se conecta directamente con la base de datos y convierte una tabla en un archivo de figuras, cuya sintaxis se muestra a continuación:

```
pgsql2shp [<opciones>] <nombre de la base de datos> <nombre de la tabla>
```

A continuación se describen las principales opciones que se le pueden especificar a través del *CLI*.

- **-d:** Escribe un archivo de figuras 3D siendo el 2D el que tiene por defecto.

- **-f <archivo>**: El valor del archivo define la ruta y nombre especifico de la salida.
- **-p <puerto>**: Define el puerto de conexión con la base de datos en función de extraer los datos.
- **-h <host>**: Define el identificador del servidor donde se encuentra publicada la base de datos.
- **-p <password>**: Define la contraseña de acceso a la base de dato especificada.
- **-u <user>**: define el identificador semántico para el usuario con los privilegios necesarios para extraer la información requerida.
- **-g <columna geometría>**: En caso de que la tabla especificada contenga varias columnas geométricas, permite seleccionar la que se va a utilizar.

El operador `&&` solo chequea las zonas de superposición de las cajas exteriores, pero se puede usar la función `truly_inside()` para obtener solo aquellas características que pasan por la caja de búsqueda. Se pueden combinar el operador `&&` para hacer un búsqueda indexada y `truly_inside()` para precisar el conjunto de resultados. Por ahora `truly_inside` solo funciona para buscar en cajas, no en cualquier tipo de geometría.

```
SELECT [COLUMN1], [COLUMN2], AsText([GEOMETRYCOLUMN]) FROM [TABLE]
WHERE [GEOM_COLUMN] && [BOX3D] AND truly_inside([GEOM_COLUMN], [BOX3D]);
```

Por otra parte la forma mas eficiente de encontrar todos los objetos que están a un radio determinado de otro, es combinando la consulta de radio con la de caja circunscrita. Esta ultima nos permite la consulta con índices espaciales, a la que después se le aplica la consulta de radio. Por ejemplo en caso que se requiera obtener todo los objetos que están a 100 metros del punto (1000 1000) seria:

```
SELECT * FROM GEOTABLE
WHERE
 GEOM && GeometryFromText('BOX3D(900 900,1100 1100)', -1)
AND
 Distance(GeometryFromText('POINT(1000 1000)',-1), GEOM) < 100;
```

Otro de las situaciones frecuentes en determinados tipos de análisis requiere llevar a cabo una reproyección de coordenadas en una consulta. Por tanto es importante tener presente que los sistemas de coordenadas fuente y destino han de estar en la tabla `ATLAL_REF_SYS`, y el objeto que se va a proyectar ha de tener como `SRID` uno de los dos valores.

```
SELECT Transform(GEOM, 4296) FROM GEOTABLE;
```

Nótese como se emplea la función `Transform` en función de llevar a cabo el objetivo planteado anteriormente. A la cual se le debe especificar por parámetro el campo geométrico y el `srid` de la nueva proyección.

## 8.5 Índices

Sin los índices cualquier búsqueda en una base de datos de gran tamaño sería interminable por tener que hacerse de forma secuencial. PostgreGIS soporta 3 tipos de índices por defecto: B-Tree, R-Tree y GIST<sup>47</sup>.

- **B-Tree**: se usan sobre datos que pueden ser ordenados sobre un eje; como por ejemplo, números, letras, fechas. Pero los datos *GIS* no pueden ordenarse sobre

---

<sup>47</sup> GIST acrónimo de Generalized Search Trees

un eje.

- **R-Trees:** divide los datos en rectángulos, subrectángulos, subsubrectángulos, etc. La implementación de *PostgreSQL* de los *R-Tree* no es tan robusta como la implementación *GIST*.
- **GIST:** Estos índices dividen los datos en “cosas que están a un lado”, “cosas que se superponen” y “cosas que están dentro”. Además soporta un amplio rango de tipos de datos, incluidos los datos GIS. PostGIS usa índices *R-Tree* sobre *GIST* para indexar los datos G/S.

### 8.5.1 GIST

GIST proporciona un Árbol de busca generalizado y una forma generalizada de indexación. Además de proporcionar indexación sobre datos GIS, GIST aumenta la velocidad de las búsquedas con toda clase de estructuras irregulares(datos espectrales, arrays de enteros,etc) las cuales no se pueden tratar con la indexación basada en B-Tree.

Cuando la tabla espacial excede unos pocos cientos de filas, y si las búsquedas no solo están basadas en atributos, es recomendable crear un índice para incrementar la velocidad de las búsquedas. La sintaxis para construir un índice sobre una columna geométrica es:

```
CREATE INDEX [nombre del índice] ON [nombre de la tabla]
 USING GIST ([campo geometrico] GIST_GEOMETRY_OPS);
```

La creación de un índice es una tarea intensiva en términos computacionales, una tabla de 1 millón de registros en una maquina a 300 MHZ de micro procesador, requiere una hora para construir un índice *GIST*. Por esto es importante que después de construir un índice forzar a *PostgreSQL* a guardar las estadísticas de las tabla, que después serian usadas para optimizar los planes de consulta: *VACUUM ANALYZE*;

Los Índices GIST tienen dos ventajas sobre los *R-Tree* implementados en *PostgreSQL*:

- GIST pueden indexar columnas con valores nulos.
- Soportan el concepto de lossiness(permite almacenar la parte importante del objeto grande en un índice) el cual es importante cuando tratamos con objetos de mas de 8K.

### 8.5.2 Empleo

Una vez construido el índice el planificador de consultas decide cuando usar el índice de forma transparente. Pero para los Índices *GIST* el planificador de consultas de *PostgreSQL* no esta optimizado y realiza búsquedas secuenciales aun teniendo el índice. Si no se están usando los Índices *GIST* en la base de datos podemos hacer dos cosas:

1. Asegurarnos de que hemos ejecutado “**VACUUM ANALYZE [tabla]**” en las tablas que nos dan problemas. *Vacuum analyze* recoge estadísticas sobre el numero y distribución de los valores en la tabla, esto ayuda al planificador de consultas a tomar la decisión de usar el índice. Es una buena costumbre ejecutar *vacuum analyze* de forma regular.
2. Si el uso de *vacuum* no funciona debemos forzar al planificador a usar la

información del índice con el comando “**SET=OFF**”. Este comando ha de usarse con moderación y solo con consultas espaciales . Una vez terminada la consulta tenemos que volver a poner a on “**ENABLE\_SEQSCAN**”, para que se use de forma normal en otras consultas. A partir de la versión 0.6 de PostGIS no es necesario forzar al planificador a usar el índice usando **ENABLE\_SEQSCAN**.

## 8.6 Funciones

A continuación se procederá a describir las principales funciones que componen el catálogo definido y aprobado por la *OpenGIS* para la extensión de *PostgreSQL* denominada como *PostGIS*.

- **AddGeometryColumn** (**varchar** dbname, **varchar** tbname, **varchar** cname, **integer** srid, **varchar** type, **integer** dimension)
  - **dbname**: Propiedad de tipo cadena que define el nombre de la base de datos.
  - **tbname**: Propiedad de tipo cadena que define el nombre de la tabla.
  - **cname**: Propiedad de tipo cadena que define el nombre de la columna.
  - **type**: Propiedad de tipo cadena que define el tipo de objeto espacial.
  - **srid**: Propiedad numérica de tipo entera que define el valor de la clave foránea que referencia SPATIAL\_REF\_SYS.
  - **dimension**: Propiedad numérica de tipo entera que define el valor de la dimensión espacial de la columna (2D o 3D).

Esta función añade una columna geométrica a una tabla existente. SRID debe ser un valor entero que referencia una valor en la tabla SPATIAL\_REF\_SYS. El tipo debe ser una cadena en mayúsculas que indica el tipo de geometría, como, POLYGON o MULTILINESTRING, etc.

- **DropGeometryColumn** (**varchar** dbname, **varchar** tbname, **varchar** cname)
  - **dbname**: Propiedad de tipo cadena que define el nombre de la base de datos.
  - **tbname**: Propiedad de tipo cadena que define el nombre de la tabla.
  - **cname**: Propiedad de tipo cadena que define el nombre de la columna.

Esta función permite eliminar una columna geométrica de una tabla espacial.

- **AsBinary(geometry geom)**
  - **geom**: Propiedad de tipo *geometry* que define la geometría en formato WKB

Esta función devuelve el valor binario de la geometría a partir de su especificación WKB de OGC, usando la codificación *endian* del servidor donde se ejecuta la base de datos.

- **Dimension(geometry geom)**
  - **geom**: Propiedad de tipo *geometry* que define la geometría.

Esta función devuelve valor 2 si la geometría es de dimensión 2D y 3 en caso de ser especificada como 3D.

- **Envelope(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Retorna un POLYGON que representa la caja circunscrita de la geometría.

- **GeometryType(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Retorna el tipo de geometría en formato de cadena, arrojando valores comprendidos en el siguiente rango [POINT / LINESTRING / POLYGON / MULTIPOLY / GEOMETRYCOLLECTION]

- **X(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Encuentra y devuelve la coordenada X del primer punto de *geometry*. Devuelve NULL si no hay puntos.

- **Y(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Encuentra y devuelve la coordenada Y del primer punto de *geometry*. Devuelve NULL si no hay puntos.

- **Z(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Encuentra y devuelve la coordenada Z del primer punto de *geometry*. Devuelve NULL si no hay puntos.

- **NumPoints(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Busca y devuelve el numero de puntos del primer linestring en la *geometry*. Devuelve NULL sino hay *linestring*.

- **PointN(geometry geom, integer point)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.
  - **point:** Propiedad numérica de tipo entera que indica la posición del punto a seleccionar.

Devuelve el punto enésimo en el primer linestring de *geometry*. Retorna valor NULL sino hay ningún *linestring*.

- **ExteriorRing(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Devuelve el circulo exterior del primer polygon en la geometry. Retorna valor nulo sino hay polígonos.

- **NumInteriorRings(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Devuelve el numero de círculos interiores del primer polígono de la geometría. Retorna valor NULL sino hay ningún polígono.

- **InteriorRingN(geometry geom, integer point)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.
  - **point:** Propiedad numérica de tipo entera que indica la posición del punto a seleccionar.

Devuelve el enésimo circulo interior del primer polígono en la *geometry*. Retorna valor *NULL* sino hay polígonos.

- **IsClosed(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Devuelve cierto si punto final es equivalente al inicial de la geometría.

- **NumGeometries(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

En caso que la propiedad *geom* tome valor *GEOMETRYCOLLECTION* devuelve el numero de geometrías que la componen. En caso contrario devuelve NULL.

- **Distance(geometry geomA, geometry geomB)**

- **geomA:** Propiedad de tipo *geometry* que define la geometría inicial denominada A.
  - **geomB:** Propiedad de tipo *geometry* que define la geometría final denominada B.

Devuelve la distancia cartesiana entre dos geometrías (*A-B*) en unidades proyectadas.

- **AsText(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Devuelve una representación textual de una geometría. Ejemplo *POLYGON(0 0,0 1,1 1,1 0,0 0)*

- **SRID(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría.

Devuelve un numero entero que es el identificador del sistema de referencia espacial de una geometría.

- **GeometryFromText(varchar geom, integer srid)**
  - **geom:** Propiedad de tipo texto que define la geometría.
  - **srid:** Propiedad numérica de tipo entera que define el valor de la clave foránea que referencia *SPATIAL\_REF\_SYS*.

Convierte un objeto de la representación textual a su valor geométrico.
- **GeomFromText(varchar geom, integer)**
  - **geom:** Propiedad de tipo texto que define la geometría.

Esta función posee un comportamiento similar a *GeometryFromText* .
- **SetSRID(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría.

Establece el valor del *SRID* de una geometría al entero dado. Usado en la construcción de cajas circunscritas para consultas.
- **EndPoint(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría.  
Devuelve un objeto punto que representa el ultimo punto en la geometría.
- **StartPoint(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría.  
Devuelve un objeto punto que representa el primer punto en la geometría.
- **Centroid(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría.  
Devuelve un punto que representa el centroide de la geometría.
- **A<&B**  
Devuelve verdadero si la caja que circscribe a A superpone o esta a la derecha de la de B.
- **A&>B**  
Devuelve verdadero si la caja que circscribe a A superpone o esta a la izquierda de la de B.
- **A<<B**  
Devuelve verdadero si la caja que circscribe a A esta estrictamente a la derecha de la de B.
- **A>>B**  
Devuelve verdadero si la caja que circscribe a A esta estrictamente a la izquierda de la de B.

- **A~B**  
Es equivalente al operador “igual que”. Compara las 2 geometrías característica por característica y si todas coinciden devuelve verdadero.
- **A~B**  
Devuelve verdadero si la caja circunscrita de A esta contendida en la de B.
- **A&B**  
Si la caja circunscrita de A se superpone a la de B devuelve Verdadero.
- **area2d(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría.  
Devuelve el área de una geometría si es un POLYGON o MULTIPOLYGON.
- **asbinary(geometry geom, 'NDR')**
  - **geom:** Propiedad de tipo *geometry* que define la geometría  
Devuelve la geometría en el formato binario de OGC, usando codificación little-endian. Se usa para sacar datos de la bd sin convertirlos a texto.
- **asbinary(geometry geom, 'XDR')**
  - **geom:** Propiedad de tipo *geometry* que define la geometría  
Devuelve la geometría en el formato binario de OGC, usando codificación big-endian. Se usa para sacar información de la base datos sin convertirla a texto.
- **box3d(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría  
Devuelve una BOX3D que representa la máxima extensión de la geometría.
- **collect(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría  
Devuelve un objeto *GEOMETRYCOLLECTION* a partir de un conjunto de geometrías. Es una función de Agregación en la terminología de PostgreSQL. Por ejemplo en caso de requerir una colección separada para cada valor distinto de ATTRCOLUMN sería de la siguiente forma:

```
SELECT COLLECT(GEOM) FROM GEOTABLE GROUP BY ATTRCOLUMN
```

- **distance\_spheroid(point pinitial, point pfinal, spheroid)**
  - **pinitial:** Propiedad de tipo *point*, describe al punto inicial para efectuar el calculo.
  - **pfinal:** Propiedad de tipo *point*, describe al punto final para efectuar el calculo.

Devuelve la distancia lineal entre dos puntos descritos por la estructura

[latitud/longitud] de un *spheroid*, para mayor comprensión ver el apartado *length\_spheroid()*.

- **extent(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría

Es una función de agregación en la terminología de *PostgreSQL*. Trabaja sobre una lista de datos, de la misma manera que las funciones *sum()* y *mean()*. Por ejemplo en caso que se requiera un *BOX3D* dando la máxima extensión a todas las características de la tabla la consulta seria de la siguiente forma:

```
SELECT EXTENT(GEOM) FROM GEOMTABLE
```

Por el contrario si la necesidad implicara obtener un resultado extendido para cada categoría.

```
SELECT EXTENT(GEOM) FROM GEOMTABLE GROUP BY CATEGORY
```

- **find\_srid(varchar container, varchar tbname, varchar cname)**

- **container:** Propiedad de tipo cadena que define el nombre de la base de datos o del esquema e que se encuentra contenida la tabla.
  - **tbname:** Propiedad de tipo cadena que define el nombre de la tabla.
  - **cname:** Propiedad de tipo cadena que define el nombre de la columna.

Devuelve el *SRID* de una columna dada buscando esta en la tabla *GEOMETRY\_COLUMNS*. Si la columna geométrica no ha sido añadida con la función *AddGeometryColumns()* no funcionará.

- **force\_collection(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría

Convierte una geometría en una *GEOMETRYCOLLECTION*. Esto se hace para simplificar la representación WKB .

- **force\_2d(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría

Fuerza la geometría a 2D así la representación de salida tendrá solo las coordenadas X e Y. Se usa porque OGC solo especifica geometrías 2D.

- **force\_3d(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría

Fuerza la geometría a 3D. Así la todas las representaciones tendrán 3 coordenadas X,Y y Z.

- **length2d(geometry geom)**

- **geom:** Propiedad de tipo *geometry* que define la geometría

Devuelve la longitud 3d de la geometría si es una linestring o multilinestring.

- **length3d(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría

Devuelve la longitud 2d de la geometría si es una linestring o multilinestring.

- **length\_spheroid(geometry geom, spheroid)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría

Calcula la longitud de una geometría en un elipsoide. Se usa si las coordenadas de la geometría están en el punto definido por la estructura [latitud/longitud]. El elipsoide es un tipo separado de la base de datos y se puede construir como:

```
SPHEROID [
 <NAME>,
 <SEMI-MAJOR AXIS>,
 <INVERSE FLATTENING>
]
```

Ejemplo:

```
SPHEROID ["GRS_1980", 6378137, 298,257222101]
```

Ejemplo de calculo:

```
SELECT LENGTH_SPHEROID(
 geometry_column,
 'SPHEROID["GRS_1980", 6378137, 298.257222101]'
)
FROM geometry_table;
```

- **length3d\_spheroid(geometry geom, spheroid)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría

Calcula la longitud de una geometría en un elipsoide, teniendo en cuenta la elevación.

- **max\_distance(linestring lsA, linestring lsB)**
  - **lsA:** Propiedad de tipo *linestring* que define el objeto inicial para comenzar a realizar la medición.
  - **lsB:** Propiedad de tipo *linestring* que define el objeto final para concluir a realizar la medición.

Devuelve la distancia mas larga entre dos *linestring*.

- **mem\_size(geometry geom)**
  - **geom:** Propiedad de tipo *geometry* que define la geometría

Esta función retorna el tamaño en bytes de la geometría.

- **npoints(geometry geom)**

- **geom:** Propiedad de tipo geometry que define la geometría

Devuelve el numero de puntos en la geometría.

- **nrings(geometry geom)**

- **geom:** Propiedad de tipo geometry que define la geometría

Si la geometría es un polígono o multipolígonos devuelve el numero de círculos de la geometría.

- **num\_sub\_objects(geometry geom)**

- **geom:** Propiedad de tipo geometry que define la geometría

Devuelve el numero de objetos almacenados en la geometría. Se usa en Multigeometrías y GEOMETRYCOLLECTIONS.

- **perimeter2d(geometry geom)**

- **geom:** Propiedad de tipo geometry que define la geometría

Devuelve el perímetro 2d de la geometría, si esa geometría es un polígono o un multipolígonos

- **perimeter3d(geometry geom)**

- **geom:** Propiedad de tipo geometry que define la geometría

Devuelve el perímetro 3d de la geometría, si esa geometría es un polígono o un multipolígonos

- **point\_inside\_circle(geometry geom, float centerCX, float centerCY, float radius)**

- **geom:** Propiedad de tipo geometry que define la geometría
- **centerCX:** Propiedad numérica de tipo coma flotante que define el valor de la longitud para el centro del circulo.
- **centerCY:** Propiedad numérica de tipo coma flotante que define el valor de la latitud para el centro del circulo.
- **radius:** Propiedad numérica de tipo coma flotante que define el valor para el radio del circulo.

Devuelve verdadero si la geometría es un punto y esta dentro del circulo.

- **postgis\_version()**

Devuelve la versión de las funciones postgis instaladas en la bases de datos.

- **summary(geometry geom)**

**geom:** Propiedad de tipo geometry que define la geometría.

Devuelve un resumen en formato de texto plano del contenido de la geometría especificada.

- **transform(geometry geom, integer srid)**

Devuelve una nueva geometría con sus coordenadas transformadas la SRID dada por el parámetro integer. SRID debe existir en la tabla SPATIAL\_REF\_SYS.

- **translate(geometry geom, float X, float Y, float Z)**

Traslada la geometría a la nueva localización usando los valores pasados como desplazamientos X,Y,Z.

- **truly\_inside(geometry geomA, geometry geomB)**

- **geomA:** Propiedad de tipo *geometry* que define la geometría denominada A.
- **geomB:** Propiedad de tipo *geometry* que define la geometría denominada B.

Devuelve verdadero si alguna parte de B esta dentro de la caja circunscrita de A.

- **xmin(box3d) / ymin(box3d) / xmin(box3d)**

Estas tres funciones devuelven la mínima coordenada de la caja circunscrita.

- **xmax(box3d) / ymax(box3d) / zmax(box3d)**

Estas tres funciones devuelven la máxima coordenada de la caja circunscrita

## 8.7 Migración de bases de datos en función del soporte de postgis

Pasos para migrar una db en función de que soporte datos espaciales sobre Postgres 8.3

sudo su postgres

createdb <db-name>

createlang plpgsql <db-name>

if postgresql-8.3

psql -d <db-name> -f /usr/share/postgresql-8.3-postgis/lwpostgis.sql

psql -d <db-name> -f /usr/share/postgresql-8.3-postgis/spatial\_ref\_sys.sql

else if postgresql-8.4

psql -d <> -U postgres -f /usr/share/postgresql/8.4/contrib/postgis.sql

psql -d <> -U postgres -f /usr/share/postgresql/8.4/contrib/spatial\_ref\_sys.sql

psql -d sadim -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql

ejemplo de empleo de esta sección refiérase al acápite xx donde se describen las configuraciones necesarias para el correcto funcionamiento del plugin postgis

## 9. Herramientas complementarias

En consecuencia a la creciente necesidad y demanda del empleo de SIG, la comunidad internacional ha desarrollado una amplia gama de utilidades en función de garantizar el soporte para la construcción de este tipo de productos, sobre todo por la complejidad de los procesos que en estos intervienen, entre las que se encuentran: *Legend*, *Msecrypt*,

*Syntax, Scalebar, Shp2img, Shptree, Shptreevis, Sortshp, Sym2img, Tile4ms, GDAL, OGR, Proj4*, etc. Teniendo en cuenta la importancia del empleo de estas, se estarán abordando algunas de las herramientas durante el desenlace de este acápite.

## 9.1 GDAL

## 9.2 OGR

## 9.3 Proj4

La librería proj4, fue creada el año 1980, esencialmente por Gerald Evenden, inicialmente para el mundo UNIX, pero con el tiempo se ha masificado para otros sistemas operativos. Por estar inserto en el mundo del Open Source, es sumamente utilizado en distintos proyectos como MapServer, GRASS, PostGIS, o las librerías OGR-GDAL, por citar algunos. Justamente MapServer utiliza esta librería para proyectar cartográficamente toda su información.

Básicamente consiste en una sencilla aplicación de líneas de comandos que reúne una serie de funciones, tales como transformación a distintas proyecciones, realizar la transformación de un sistema de referencia a otro, cálculos de problema directo e inverso, entregando toda la información arrojada por el desarrollo de cálculos estandarizados en la ISO 19100 y 19111. Al momento de liberar su código, esta aplicación ha desarrollado mayores funcionalidades en su bases de comandos, de hecho se ha integrado como parte fundamental de numerosas aplicaciones que requieren funciones cartográficas de proyección. Por el momento existe una versión precompilada para Windows, así como versión para Debian 14 GNU/Linux.

Fundamentalmente la librería Proj4 se divide en tres grandes comandos, *proj*, *cs2cs*, *geod*, teniendo cada uno de ellos finalidades específicas.

### 9.3.1 Reproyección de coordenadas georeferenciadas

El comando ***proj*** esta destinado para realizar cambios de proyección de coordenadas, entregando también opcionalmente una serie de variaciones ligadas estos tipos de cálculos. Estas modificaciones se podrán efectuar utilizando correctamente los siguientes parámetros:

- I** Permite especificar la proyección inversa a las coordenadas inicialmente ingresadas. Sin embargo puede ser redundante cuando se está utilizando el comando *invproj*.
- r** La función de éste es poder invertir el orden de las coordenadas de entrada, pudiendo ingresar la latitud y longitud o longitud y latitud, solo separado por espacio.
- s** Posibilita alterar el orden de salida de las coordenadas, permitiendo entregar el orden invertido de las coordenadas iniciales.
- f** Básicamente se utiliza para traspasar de coordenadas geodésicas en formatos

de grados minutos y segundos, a grados decimales.

- le Entrega un listado de todos los elipsoides que se pueden seleccionar con +ellps, comando interno de proj4
- v Muestra un listado de los parámetros cartográfico probados y usados para ser utilizados por el programa.
- lu Arroja un listado de todas las unidades que se pueden seleccionar con +units, comando interno de proj4

### 9.3.2 Transformación de sistema de referencia de coordenadas

El comando **cs2cs** cumple la función de ejecutar el traspaso de un sistema de referencia de coordenadas a otro. Este tipo de cálculos matemáticos, así como todos los ejecutados en la librería, han sido desarrollados de acuerdo a lo establecido en la ISO19111, se debe tener en cuenta, que de acuerdo a la versión de la librería proj4 que se utilice, dependerá la cantidad posible de traspasos de un sistema a otro a realizar, debido a que las versiones más recientes han alcanzado sustentables mejoras en comparación con sus versiones iniciales.

En la figura que se muestra a continuacion se expondrá un ejemplo práctico que represente la funcionalidad de la librería Proj4, en la cual se ingresarán coordenadas en el sistema provisional para Sudamérica (*PSAD-56*) en proyección Universal Transversal de Mercator, para que la librería lo transforme a *WGS84* y lo entregue en coordenadas geográficas en grados decimales.

```
D:\>cs2cs +proj=utm +zone=19 +south +ellps=intl +towgs84=-270,183,-390 +unit=m +
to +proj=latlong +datum=WGS84 -f "%.8f"
394838.11 6360304.09
-70.12631493 -32.89338359 214.322
```

**Ilustración 95** Prototipo de cambio de sistema de referencia

La precisión que entrega este cambio en el sistema de referencia es +- 5 metros. Cabe destacar que para un correcto funcionamiento en cuanto a la exactitud de este cálculo, se deben ingresar los parámetros de traslación entre un sistema y otro para así poder obtener correctos niveles de exactitud. Los mencionados parámetros fueron creados por el científico ruso Sergui Molodensky y explicado en su debido momento. Estos parámetros deben ser modificados para traspasar distintos sistemas de referencia, pues son únicos para cada tipo de traspaso.

### 9.3.3 Cálculos de Azimut y distancia de problema inverso

Otra herramienta que es posible utilizar dentro de la librería *proj4*, es **geod**, la cual tiene por finalidad ejecutar cálculos geodésicos, los cuales son problema directo o problema inverso, la precisión entregada por esta librería, esta en el orden de +-0.0004 segundos, adoptando todos las condiciones impuestas en la ISO 19100.

En la Fig. 3.3 se realiza un ejemplo de cálculo del problema inverso, el cual entrega distancia Geodésicas y el respectivo azimut

```
C:\ms4w\proj\bin>geod +ellps=clrk66 -I +units=m
42d15'N 71d07'W 45d31'N 123d41'W
-66d31'50.141" 75d39'13.083" 4164192.708
```

**Ilustración 96** Ejecución de calculo de problema inverso

## 9.4 Shp2img

Esta herramienta permite generar una salida muy similar a la que brindaría el servidor de mapas Mapserver tomando como base un *mapfile*. Para utilizarla se debe ejecutar desde el CLI especificándole los parámetros de invocación con una sintaxis similar a la que se muestra a continuación:

```
shp2img -m mapfile [-o image] [-e minx miny maxx maxy] [-s sizex sizey]
[-l "layer1 [layers2...]" [-i format]
[-all_debug n] [-map_debug n] [-layer_debug n] [-p n] [-c n] [-d layername datavalue]
```

A continuación se procede a describir algunas de las opciones mas utilizadas:

- **-m <mapfile>**: Define la ruta del fichero *mapfile* necesario para generar la salida.
- **-i <format>**: Permite redefinir el valor la propiedad *imagetype* del *mapfile* en función de generar la salida con el formato especificado.
- **-o <image>**: Define el fichero de salida.
- **-e <minx miny maxx maxy>**: Permite redefinir el valor de la propiedad *extend* del *mapfile*.
- **-s <sizex sizey>**: Define las dimensiones de la imagen generada.
- **-l <layers>**: Define la capa o el listado de capas que serán habilitadas en el mapa, en caso de ser mas de una, deberán estar delimitadas por el carácter de comillas dobles ("") y separadas por un espacio.
- **-all\_debug <n>**: Establece el nivel de depuración para el mapa y todas las capas.
- **-map\_debug <n>**: Establece el nivel de depuración para el mapa.
- **-layer\_debug <n>**: Establece el nivel de depuración para la capa.
- **-c <n>**: Permite dibujar el mapa *n* veces.
- **-p <n>**: Pausa durante *n* segundos después de leer el mapa.

## 9.5 Legend

Propósito: Crear una leyenda de un Mapfile. La salida de es PNG o GIF dependiendo que versión de la librería GD que es usada.

Sintaxis: **legend** [mapfile] [output image]

## 9.6 Scalebar

Propósito: Crear una scalebar (barra de escala) para un Mapfile. La salidas de esto pueden ser es PNG o GIF dependiendo que versión de la librería GD usada.

Sintaxis: **scalebar** [mapfile] [output image]

## 9.7 Sortshp

Propósito: Ordenar un shapefile en una columna basado en un orden ascendente o descendente. Soporta columnas integer, double y tipos de columna string. Usando para priorizar formar al reenderezar o etiquetar.

Sintaxis: **sortshp** [infile] [outfile] [item] [asendign/descendig].

## 9.8 Sym2imp

Propósito: Crear un graphic dump de un symbol file. La salida pueden ser PNG o GIF dependiendo de la versión de la librería GD.

Sintaxis: **sym2img** [symbolfile] [outfile].

## 9.9 Shptree

Propósito: Crear un quadtree basado en un índice especial de un shapfile. La profundidad del árbol es calculada para cada nodo del árbol conteniendo 8 shapefile. No use el valor por default para los archivos de puntos, un valor entre 6 y 10 parece ser el adecuado para trabajar. Su medida puede variar y usted necesitara hacer algunas pruebas con estas.

Sintaxis : **shptree** [shpfile] [depth]

## 9.10 Tile4ms

Propósito: Crear un cuadro índice en el shape file para trabajar con Mapserver y las características de tileindex. EL programa crea un shapefile de forma rectangular para la extensión de todos los shapefiles listado en el [metafile](un shape nombrado por linea) y la Base de Datos con el nombre de archivo para cada tile shape en una columna llamada LOCATION y que es requerido por mapserv.exe.

Sintaxis: **tile4ms** [metafile] [tilefile] [tilepathonlytilepath] esta ultima opcion es solo una etiqueta opcional que especifica que solo el path de el shape file podría ser almacenado en al etiqueta LOCATION.

## Glosario de términos

- **ACAXIA:** Sistema integral de gestión de seguridad.
- **DPA:** División Político Administrativa.
- **ERS:** Especificación de requisito de software.
- **GeneSIG:** Plataforma Soberana para el desarrollo de Sistemas de Información geográfica.
- **GIS:** Abreviatura en inglés de Sistemas de Información Geográfico.
- **IOC:** Patrón de integración inversión de control.
- **Zeolides:** Marco de trabajo para el desarrollo de aplicaciones web.

## Bibliografía

- **Campocamp SA. 2008.** CartoWeb. [En línea] 2008. <http://cartoweb.org/>.
- **OSGeo. 2011.** PostGIS. [En línea] 2011. <http://postgis.refractions.net/>.
- **The PHP Group. 2001.** PEAR. [En línea] 2001. <http://pear.php.net/>.
- —. 2001. PHP. [En línea] 2001. <http://www.php.net/>.
- **Yoenis Pantoja Zaldívar, Lidisy Hernández Montero.** 2010. Documento Visión v1.0. Habana : UCI, 2010.
- **DM Solutions Group, MapGears, and Gateway Geomatics.** Map tools. [En línea] [Citado el: 27 de 5 de 2012.] <http://maptools.org/fgs/>.