CURSO DE EXPERTO EN DESARROLLO Y GESTIÓN DE SISTEMAS DE INFORMACIÓN GEOGRÁFICA (31-EX-75)

Estudios propios de la Universidad de Cantabria

MATERIALES DOCENTES

Materia Nº 4 – Difusión y publicación de resultados

11-M4.2 – Desarrollo de aplicaciones geográficas web: OpenLayers

MATERIALES DE APOYO

Profesor:

VÍCTOR VELARDE GUTIÉRREZ

Enero 2011 – 1ª edición / victor.velarde@gmail.com

Índice de contenidos

Índice de ilustraciones

Presentación

Objetivos de la asignatura

- Conocer qué tipo de nuevas aplicaciones geográficas existen en la web y en qué se diferencian de las aplicaciones SIG tradicionales.
- Adquirir los conocimientos básicos sobre Internet y los mecanismos informáticos que hacen posible el funcionamiento de OpenLayers y, en general, las aplicaciones web.
- Poder utilizar HTML, Javascript y CSS a nivel básico para realizar un *mashup* con OpenLayers y diversos proveedores de contenidos.
- Conocer la biblioteca OpenLayers y sus capacidades generales para el desarrollo de páginas web.

Planteamiento de las clases

La asignatura tiene un enfoque fundamentalmente práctico y como tal se realizarán varios ***ejercicios con el ordenador*** durante el curso. Para cada ejercicio se proporcionará un documento-guía y los ficheros necesarios y a su finalización se proporcionarán las soluciones-tipo.

Durante las clases se alternarán los ejercicios con la ***presentación de contenidos*** básicos. En éste documento se recogen dichos contenidos, organizados en torno a 3 áreas: las nuevas aplicaciones geográficas en la web, fundamentos de programación web y funciones básicas de OpenLayers. Además incluiremos otras referencias para profundizar en el aprendizaje, en forma de enlaces a lecturas y ejercicios complementarios.

Finalmente el alumno realizará tras finalizar las clases presenciales un ***ejercicio práctico final*** que remitirá al profesor por correo electrónico, en el plazo que le sea indicado. Esta práctica será tenida en cuenta en la evaluación de la asignatura con un 70% de la nota final. El 30% restante de la nota se obtendrá con el trabajo continuo realizado en el aula.

1. Nuevas aplicaciones geográficas en la Web.

En este apartado nuestro objetivo es proporcionar una descripción del marco general en el que se encuadra la asignatura: la realización de aplicaciones geográficas web con *OpenLayers*.

Primero presentaremos los actuales planteamientos para el manejo de información geográfica, los cuales están centrados en los servicios web y han supuesto un importante cambio en la comunidad SIG más tradicional. Se introducirán algunos conceptos relevantes como *Neogeografía* y *mashup*, así como el importante papel que ocupan las librerías de programación como GoogleMaps y OpenLayers en estos fenómenos.

A continuación comentaremos la importancia de estas nuevas tendencias y herramientas como la programación para los profesionales que trabajan con información geográfica, así como los nuevos retos a los que estos se tienen que enfrentar.

1.1. La revolución de las tecnologías geoespaciales y la Neogeografía

En los últimos años se está produciendo una evolución muy notable de los SIG y las tecnologías geoespaciales en general. Antes sólo eran usadas por un reducido grupo de científicos y profesionales de la gestión del territorio, pero ahora su uso es mucho más generalizado y variado.

Hasta hace pocos años, los **Sistemas de Información Geográfica** se identificaban netamente con un software concreto (como p.ej. ArcGIS o GRASS), que estaba instalado en un ordenador o una estación de trabajo muy potente, probablemente en una administración pública, una universidad o una gran firma de ingeniería. La inversión en recursos hardware, licencias, formación del personal... era tan alta que los únicos proyectos en los que se podían dar el lujo de utilizarlos eran aquellos con un amplio presupuesto, como por ejemplo un inventario nacional de recursos forestales, el Catastro o el diseño y mantenimiento de redes en las compañías eléctricas. Era inviable para un no profesional acceder a la información y a las herramientas informáticas implicadas.

Sin embargo, hoy en día es muy común que cualquiera tenga instalado en su PC *GoogleEarth* y planifique con él un viaje de ocio, use un navegador GPS con cartografía de detalle en su coche o que por ejemplo un grupo de comerciantes publique dónde están sus negocios en la página web de su asociación.

El trasfondo de estos proyectos sigue siendo el mismo: el manejo de información con componente espacial proporciona a su usuario un conocimiento muy poderoso para la toma de decisiones. Pero las

herramientas, el perfil de los usuarios, el hardware... todo lo demás ha cambiado, diversificándose de manera intensa.

Esto es lo que ha llevado a algunos expertos a reinterpretar el neologismo de *Neogeografía*. La **Neogeografía**, según su concepción más reciente, se puede definir como el conjunto de *"herramientas y técnicas geográficas empleadas para actividades personales o por un grupo de usuarios no expertos, para uso informal no analítico"*[1]. Es evidente cómo en los últimos años se ha generalizado el uso intensivo de información geográfica a través de la tecnología y se ha 'democratizado' su uso, dejando de ser una valiosa información recluida en silos custodiados por expertos.

Ilustración 1: Alfabeto realizado con lugares reales en *GoogleMaps*

Fuente: http://goo.gl/SLwVZ

Los tres factores fundamentales que han hecho posible esta evolución/revolución geográfica son:

1. *la nueva cartografía* – se dio el gran paso con la aparición de la cartografía de *Google* en 2005. Una base cartográfica gratuita, de ágil consulta, cobertura mundial y crecientemente detallada. Primero fue *GoogleMaps* en las páginas web y luego se introdujo la dimensión 3D en los escritorios con *GoogleEarth*[2], y con ambas la compañía

1 (2007) A. TURNER: http://highearthorbit.com/neogeography-towards-a-definition/

2 *GoogleMaps* fue anunciado por primera vez el 8 de febrero del 2005 y estuvo en su fase beta 6 meses [...]. Fue ya en junio del 2005 cuando Google lanzó su API [...]. Las fuentes de los mapas disponibles son principalmente satélites y aviones

estadounidense ha revolucionado el panorama mundial de lo geográfico. De manera paralela y complementaria, se ha ido abriendo paso la tendencia de los organismos públicos a ir liberando parte de sus datos geográficos a través de servicios OGC (WMS, WFS...).

2. *la expansión de los GPS* - el abaratamiento de los chips de GPS ha permitido que las capacidades de posicionamiento estén integradas en aparatos accesibles para los no profesionales; GPS simples, *GPS-loggers*, teléfonos móviles, navegadores, cámaras fotográficas... todos estos dispositivos han agilizado enormemente la captura de información georreferenciada (posición de personas y vehículos, trazado de rutas...).

Ilustración 2: Mapamundi con logos de servicios sociales

Fuente: http://www.appappeal.com/web-2-0-application-world-mosaic/

3. *el avance de Internet y la Web 2.0* – actualmente hay un mayor acceso a las redes de información: las personas tienen cada vez conexiones más rápidas, en más lugares y tienen herramientas más sencillas y potentes para explotarlas. Por otro lado, frente a la primera etapa de Internet, caracterizada por páginas con contenidos estáticos, poco interactivas y publicadas por expertos en informática hoy se impone lo que se ha dado en llamar la *Web 2.0:* un Internet

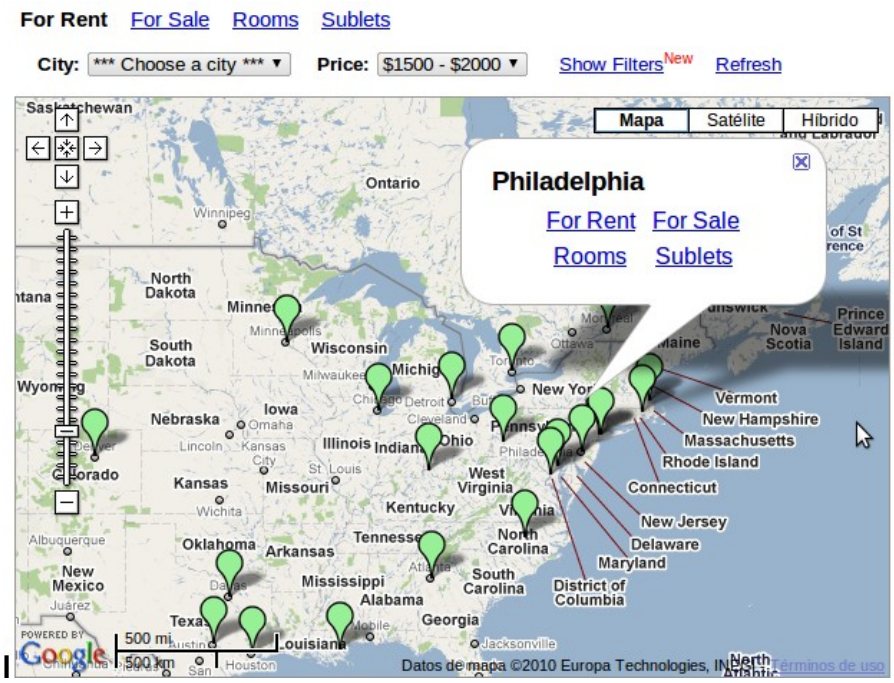
aunque también se vale de mapas digitalizados de compañías como TeleAtlas y EarthSat. *Google* asegura que su información no tiene más de tres años. "*Un poco de historia y curiosidades de Google Maps/Earth"* (en http://www.tufuncion.com/google-maps-earth).

caracterizado por la facilidad para publicar contenidos (fotos, vídeos, comentarios sobre lugares, documentos...) y los sitios/servicios de generación colectiva de información. Es la web de los blogs, wikis, los formatos de sindicación de contenidos (RSS, ATOM), de proyectos tan conocidos como Wikipedia, Youtube y las todopoderosas redes sociales como Facebook, Tuenti o Twitter. Precisamente la creciente mezcla de *lo social y lo geográfico* está produciendo avances muy interesantes como la faceta espacial de Twitter, FourSquares, Facebook Places...

1.2. Los mashups, un nuevo estilo de combinar información

La cristalización de las nuevas tendencias en información geográfica se produce en lo que se ha dado en denominar **mashups**. Un *mashup* (literalmente *puré o mezcla de elementos*) es una página web híbrida, en la que se mezclan dos o más componentes de origen externo cuya integración aporta valor añadido. Por ejemplo un *mashup* sería una página web que muestra sobre un mapa las previsiones meteorológicas de una comunidad autónoma, tomando los servicios de *GoogleMaps* y los datos del tiempo de un servicio del Ministerio de Medio Ambiente.

Ilustración 3: El primer mashup



Fuente: http://www.housingmaps.com/

El primer *mashup* publicado fue la web *Housing maps*, elaborada por Paul Rademacher en el 2005, en el que se muestra sobre los mapas de Google los pisos en alquiler o venta que aparecen en otra página web llamada *Craiglist.* Desde entonces hasta ahora se han programado miles de mashups, de toda temática y estilo, y aunque no todos poseen un mapa como elemento integrador la gran mayoría sí son 'geomashups'[3].

Los elementos que hacen posible un *mashup* son:

(1) *los servicios proporcionados por terceros*. Entran en esta categoría los servicios web de empresas y organismos públicos que son accesibles públicamente mediante APIs[4] y que ofrecen recursos RSS, JSON o XML[5]... pero también métodos menos 'ortodoxos' de captura de información desde páginas web, como p.ej. el *screen scraping.* Algunos de los servicios más comunes son:

- el API de mapas de GoogleMaps.
- el API de fotos de Flickr.
- el API de vídeos de Youtube.
- el API de microblogging de Twitter.
- el API de venta de productos Amazon.
- el API social de Facebook.
- el API de la enciclopedia Wikipedia

Un sitio web interesante para descubrir servicios utilizados en diversos mashups es *The Programmable Web*: http://www.programmableweb.com/

Para el manejo de información geográfica tenemos también muchos APIs específicos como BingMaps, YahooMaps, Geonames o

3 Una buena dirección para observar numerosos mashups con mapas es *GoogleMaps Manía:* http://googlemapsmania.blogspot.com/

4 API (*Application Programming Interface)* o Interfaz de programación de aplicaciones. Este término hace referencia al conjunto de funciones proporcionado por un software que tiene disponibles el programador cuando trabaja con él para construir una aplicación. Generalmente junto al listado estricto de los métodos el API incluye recursos complementarios como ficheros de ayuda, ejemplos...

5 Hay muchos formatos para obtener los datos desde estos servicios, como veremos en los ejercicios de la asignatura. Algunos usuales son RSS=Really Simple Syndication, JSON=JavaScript Object Notation o XML=eXtensible Markup. Language.

CloudMade.... de hecho, la importancia creciente del factor espacial ha llevado a que se extienda el término ***Geoweb***.

(2) *la programación de la página web.* El creador toma los servicios previos y los consulta, manipula, compara o combina para generar nuevos datos desde su web, valiéndose de la programación, plantillas u otras herramientas. Lo más habitual es el uso de tecnologías web estándar como HTML y en gran medida Javascript. La manera de consumir los servicios suele ser relativamente sencilla y estar bien documentada en las páginas web de los productores y por ello un gran número de personas, muchos sin ser profesionales informáticos pero sí innovadores e inquietos, se han volcado en el desarrollo de *mashups*.

(3) *la ejecución de la página en el navegador.* Una vez publicada la web, el usuario se conecta a Internet y, a través de software como Firefox, Chrome o Internet Explorer, se ponen en valor las funciones programadas por el creador. Generalmente los *mashups* se centran especialmente en el usuario 'de a pie', con interfaces sencillos y visualmente atractivos.

1.3. Cruce de caminos: la Neogeografía y los SIG

En la actualidad hay un debate vivo acerca de cómo los cambios representados por la Neogeografía afectan a la comunidad SIG más tradicional. Existen posiciones diversas al respecto, desde proclamas provocadoras que rezan cómo la "*Neogeografía ha matado a los SIG*[6], hasta grupos de profesionales SIG que viven al margen de las nuevas tendencias.

Ilustración 4: SIG tradicional frente a Neogeografía

SIG tradicional	Neogeografía
Geografía 'tradicional' / académica	Decisiones espaciales / ocio
Programa complejo de escritorio	*Mashup*
Experto en análisis del territorio	Ciudadano común
Gabinete / departamento	Web 2.0 / Red social
Software propietario / SHP	Servicios web gratuitos / KML

6 Ver la interesante presentación de A. TURNER en Slideshare: http://www.slideshare.net/ajturner/how-neogeography-killed-gis

Lo que sí es evidente es que los nuevos enfoques de manejo de información geográfica están triunfando porque están *centrados en el usuario*, por encima del énfasis en las herramientas y se podría decir que los SIG han sido empujados fuera de su 'zona cómoda' y deben adaptarse a esa realidad si quieren evolucionar.

Desde una perspectiva integradora, más que una división artificial entre los dos enfoques existen influencias mutuas: los usuarios no profesionales evolucionan poco a poco hacia análisis espaciales más complejos y los responsables de SIG comparten cada vez más sus datos mediante servicios, incorporan en sus flujos nuevos formatos como el KML o se benefician de los mapas de Google como base para mostrar los suyos.

Esto nos deja con un panorama que ha sido denominado por algunos **SIG 2.0**, una reformulación dentro de la comunidad SIG impregnada de los principios de la Neogeografía[7].

En este contexto, un profesional puede verse involucrado en proyectos que manejan información geográfica de manera muy diversa, dentro un amplio *espectro de las tecnologías de la información geográfica.* De acuerdo a su perfil, los nuevos retos a los que se enfrenta variarán:

1. Administrador SIG. Presente sólo en organizaciones con un número medio o grande de empleados, con un SIG departamental o corporativo. Como responsable del sistema hoy en día habrá de preocuparse por cuestiones como:

- ¿qué sistema de licencias de software es más ventajoso para mi organización?
- ¿proporcionamos acceso a nuestros datos solo a nuestros usuarios internos o lo hacemos de manera abierta y vía Internet al público general?
- ¿cómo lo realizamos de manera segura y controlada?
- ¿montamos toda la infraestructura hardware bajo nuestros servidores o contratamos servicios 'en la nube'?
- ¿desarrollamos nuestros servicios de mapa base o usamos GoogleMaps?
- ¿debemos implementar algún estándar oficial -WMS, WFS...- u oficioso -compatible con GoogleMaps-?
- ¿cómo conseguimos el mejor rendimiento y maximizar el valor de nuestra información geográfica?

7 Para profundizar más detalles sobre el concepto SIG 2.0 ver la presentación http://www.slideshare.net/jscampbell1/gis-20-and-neogeography

2. Desarrollador SIG. Con un perfil técnico, ha de ser capaz de desarrollar sistemas que cumplan los requisitos funcionales y no funcionales establecidos por el Administrador SIG o el gerente de su área. En su día a día se pregunta sobre aspectos como:

- ¿cómo utilizar el API de Google para utilizar su geocoding gratuito dentro de mi aplicación?
- ¿cómo programo un procedimiento para exportar los datos de mi base de datos a formato KML y pintarlos al vuelo sobre *OpenStreetMaps*?
- ¿qué librerías o herramientas me permiten trabajar de manera más eficiente en mi proyecto de software?
- ¿con qué sistema de coordenadas genero la caché de mi mapa para que se pueda solapar sobre BingMaps?
- ¿cómo genero un portal web de contenidos para mi empresa con un apartado de mapa donde pueda editar ciertas capas vinculadas a un WFS?

3. Usuario SIG. Es el bloque más diverso, pudiendo incluir desde usuarios intensivos de información geográfica, trabajando dentro de una organización con un SIG corporativo, a usuarios más ocasionales, del tipo de un profesional *freelance* de la consultoría ambiental. De acuerdo a su entorno profesional podrá delegar el desarrollo de aplicaciones SIG a otros perfiles más especializados (desarrolladores SIG) o bien iniciar él mismo proyectos de publicación web del tipo de un mashup, formándose en las herramientas y APIs necesarios.

- ¿qué proyectos con información urbanística sobre un mapa municipal encuentro en la web y cómo puedo hacer algo similar?
- ¿con qué software de escritorio convierto mis ficheros .shp para publicarlos luego en Internet?
- ¿qué servicios de mapas publica el Ministerio de Medio Ambiente y cómo los incorporo dentro de Google Earth?
- ¿cómo muestro sobre un mapa en la web mis puntos de interés, con información multimedia georreferenciada como vídeos y fotos?

1.4. Nuevos usuarios, nuevos programadores y OpenLayers

Actualmente las herramientas para el manejo de información geográfica son muy variadas: software SIG de escritorio, para móvil, páginas web con

servicios... En este contexto y ante unas necesidades de información espacial concretas nuestro primer paso debe ser buscar la solución informática más eficiente. Si esta solución de software ya ha sido desarrollada, sea de pago o gratuita, generalmente adoptarla y ser su usuario será la mejor opción.

Si nuestras necesidades no están cubiertas por el software existente, o el que hemos encontrado no nos satisface plenamente, entonces es el momento de considerar el desarrollo de software y utilizar la programación, para lograr una solución a medida. Los programadores requieren de una serie de conocimientos técnicos y especialmente de mucha práctica. Y para mejorar en esta labor es importante la documentación, los ejemplos, las herramientas y especialmente el trabajo previo de otros programadores en forma de componentes software especializados, como la librería OpenLayers.

1.4.1. El perfil del programador de SIG

Para proyectos SIG corporativos, a partir de cierto tamaño y complejidad, es recomendable contar con equipos multidisciplinares e imprescindible una buena gestión. Diseñadores, programadores, expertos temáticos, gerentes, futuros usuarios... todos ellos deben colaborar y alinear sus intereses y capacidades técnicas para que el proyecto salga adelante. Un buen resultado solo será posible siguiendo una metodología adecuada de desarrollo informático[8], para que así el trabajo se ajuste al presupuesto y los plazos establecidos.

Sin embargo para proyectos menos complejos seguramente nuestro presupuesto no nos permite tal cantidad y especialización del personal y, en lugar de tener un departamento SIG con varios profesionales, nos encontramos sólo con un usuario avanzado. Este usuario, con una formación básica en programación y los recursos disponibles en Internet (documentación, foros, ejemplos, lista de correo...), puede realizar proyectos SIG para la web muy interesantes de manera autónoma. Evidentemente con una formación técnica más profunda el usuario puede si lo desea realizar otras labores propias de un administrador o programador SIG y participar en proyectos de mayor envergadura.

El perfil de un profesional usuario avanzado / programador SIG suele incluir capacidades en las siguientes áreas:

a) Software SIG de escritorio.- el software SIG clásico, del tipo de ArcGIS, gvSIG, MapInfo o Geomedia. El usuario avanzado lo utilizará no sólo por sus capacidades de análisis más potentes sino también

8 Aunque existe numerosa literatura sobre metodologías de desarrollo, nos permitimos aconsejar siempre que sea posible la utilización de las llamadas 'metodologías ágiles', como p.ej. Scrum: http://es.wikipedia.org/wiki/Scrum

como herramienta para administrar sus datos, consumir servicios disponibles en servidores propios o externos y automatizar flujos de procesamiento de datos (p.ej. con el ModelBuilder de Esri).

b) Software SIG para servidor.- El usuario, especialmente si desarrolla labores de administración dentro de su organización, necesitará conocimientos de tecnologías para:

- El almacenamiento de datos espaciales (PostGIS, ArcSDE, Oracle Spatial...).
- La publicación de mapas, fenómenos o coberturas (WMS, WFS, WCS...) como p.ej. MapServer, Geoserver o ArcGIS Server.
- La catalogación y búsqueda de información (metadatos con CatMDEdit, Geonetwork...).

c) Sistemas de captura / georreferenciación de información.- El manejo de aparatos GPS, herramientas de geocoding online, el *geotagging* de documentos, fotos, videos... hacen posible al usuario capturar sus datos y producir información espacial utilizable.

d) APIs de mapa y otros servicios web.- Si desarrolla proyectos para la web habrá de conocer servicios de mapa tales como GoogleMaps, BingMaps, OpenStreetMap... Conocer otros servicios complementarios (Panoramio, Twitter, etc.) le permitirá añadir nuevas funciones a sus proyectos mediante integración con redes sociales o por ejemplo elementos multimedia.

e) Librerías SIG y herramientas de programación.- Las librerías de software SIG como OpenLayers son componentes reutilizables que nos permiten una programación a más alto nivel y el acceso más sencillo a las operaciones habituales. Los editores y otras herramientas, como los entornos integrados de programación (IDEs), facilitan usar esas librerías y desarrollar con mayor facilidad nuestras aplicaciones.

1.4.2. ¿Por qué usar OpenLayers?

La aparición de *GoogleMaps,* y sobre todo de la manera de utilizarlo gratuitamente en otras páginas mediante programación, ha revolucionado definitivamente el panorama geoespacial.

Los puntos más destacados del proyecto de GoogleMaps son:

- El buen diseño y documentación de su API[9].

9 El sitio oficial de la familia de APIs de GoogleMaps es http://code.google.com/intl/es-ES/apis/maps/index.html

- La cobertura, detalle y velocidad de sus mapas. La velocidad 'google', soportada por una infraestructura hardware inmensa, se ha convertido en la velocidad estándar requerida por cualquier usuario.
- La amplitud de la comunidad generada a su alrededor (con abundantes ejemplos en páginas, foros y blogs).
- Su creciente capacidad de integración y sinergia con otros servicios y tecnologías (del propio Google[10] o de terceros).

Estos factores lo han convertido en la primera opción de manejo de información espacial para muchos individuos y organizaciones en todo el mundo. Entonces ¿por qué utilizar otra opción como OpenLayers?

Ilustración 5: Ventajas de OpenLayers frente a GoogleMaps

OpenLayers	GoogleMaps
Mayor flexibilidad para mapa base: GoogleMaps, BingMaps, OpenStreetMap, WMS...	Sólo capas base propias (callejero, satélite, híbrido y relieve).[11]
Implementación estándares OGC (WMS, WFS...)	Tecnología propia de Google
Mejor manejo de entidades vectoriales y estilos.	-
OpenSource, bajo osGeo y con licencia BSD[12]	Gratuito pero no abierto, con código fuente no accesible.
Sinergias otros software libre como Geoserver.	-

Teniendo en cuenta estos factores, en cada proyecto ha de elegirse la mejor opción, sopesando además nuestros conocimientos, objetivos y cultura organizativa.

10 Productos como Google Web Kit -GWT-, Google Earth, Fusion Tables, Google Transit...

11 El API de GoogleMaps admite, por primera vez con la versión v3, la incorporación de tipos de mapa personalizados (*Custom Map Types*), con un orígen distinto del propio google, como OpenStreetMaps: http://blog-idee.blogspot.com/2010/09/acceso-openstreetmap-desde-google-maps.html

12 Licencia Berkeley Software Distribution: http://es.wikipedia.org/wiki/Licencia_BSD

2. Fundamentos de programación para OpenLayers

Enfocándonos en un usuario con interés en avanzar en el campo de la programación SIG para la web, en este apartado incluimos algunos temas que consideramos importante conocer antes de introducir la librería OpenLayers.

El primero de los apartados consistirá en un rápido vistazo a los mecanismos técnicos principales que han permitido el fenómeno imparable de Internet: los protocolos *TCP/IP* y *HTTP*.

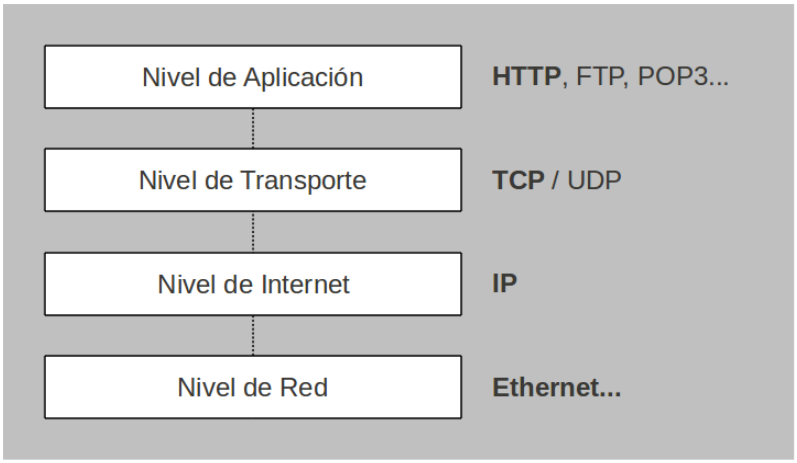
A continuación se realizará una presentación de los lenguajes imprescindibles para trabajar con OpenLayers: *HTML, CSS y Javascript*, recogiendo de ellos un subconjunto básico pero suficiente para comenzar a programar con la librería y algunas referencias complementarias para profundizar en su manejo.

2.1. Las bases técnicas de Internet: TCP/IP y HTTP

La mayoría de los usuarios están familiarizados con Internet y lo utilizan a diario para sus labores cotidianas: navegación, correo, contactos en redes sociales... La base técnica que hace posible estas funciones normalmente es 'transparente', permaneciendo en un discreto segundo plano. Sin embargo, desde el punto de vista de un programador SIG es útil conocer cuáles son los mecanismos de más bajo nivel involucrados en las aplicaciones web.

La base técnica de Internet son un conjunto de *protocolos de red* que permiten la comunicación entre equipos informáticos, combinando sus capacidades de procesamiento para la resolución de problemas.

Ilustración 6: Niveles del protocolo TCP/IP



Estas normas se denominan habitualmente ***pila de protocolos TCP/IP*** y son un estándar sólido y bien documentado, compartido por toda la comunidad de fabricantes de software y hardware. Estos protocolos también

se utilizan para hacer funcionar aplicaciones en redes de sistemas locales, denominadas *intranets.*

En la pila existen varios protocolos, cada uno con una responsabilidad específica, pero todos ellos trabajando de manera conjunta y coordinada para permitir que las aplicaciones puedan comunicarse sobre la red. El nivel más alto es el más abstracto y próximo al usuario, mientras que el nivel más bajo es el que resuelve la transmisión del flujo de bytes a través de los componentes físicos de la red. Para la transmisión de cada bloque de información un protocolo envuelve los datos en un paquete o trama y lo transmite al protocolo inmediatamente inferior; cada uno añade una porción de información mediante una cabecera y ello permite resolver todas las responsabilidades necesarias (direccionamiento, reenvío de información pérdida...).

Los protocolos de la pila más importantes son los dos que le dan nombre: TCP e IP. El protocolo TCP (*Transmission Control Protocol*) permite que los paquetes de información lleguen con fiabilidad a su destino y la información viaje entre equipos distantes y diversos. Asociado al TCP aparece el concepto de *puerto:* una valor numérico del 0 al 65534 que identifica un punto de conexión por red de una aplicación cliente con una aplicación servidora[13].

Por su parte, el protocolo IP (*Internet Protocol*) se utiliza para identificar los equipos y reconocer redes, routers y ordenadores concretos. Una dirección IP (en la versión más extendida actualmente v4), se compone de 32 bits del tipo 74.125.230.80 (que se corresponde con www.google.es[14]). Cada PC tendrá una dirección IP por cada tarjeta de red configurada.

Por su parte el ***protocolo HTTP***, cuyas siglas responden a Hyper Text Transfer Protocol o *Protocolo de Transferencia de Hipertexto,* constituye el medio básico para navegar por páginas web. Sobre las redes TCP/IP y, en general, sobre Internet, pueden ejecutarse muchos tipos de aplicaciones pero las que corren bajo HTTP son sin duda las más abundantes.

HTTP está orientado a recursos. Un recurso, identificado mediante una dirección URL (lo que comúnmente conocemos como un 'enlace'), es un elemento variado como p.ej. un fichero de HTML, una imagen, un fichero de audio o una consulta a un programa.

13 Existen 3 rangos de puertos: los primeros 1024 son *puertos reservados* para el uso del sistema operativo y las aplicaciones más extendidas (p.ej. HTTP: 80, FTP: 21, POP3: 110...). El rango siguiente, del 1024 al 49151 son los *puertos registrados* y los podemos usar en una aplicación servidor propia. De 49152 al 65535 son *puertos dinámicos y/o privados*. Estos puertos pueden ser usados por TCP o por UDP (otro protocolo de transporte pero que no garantiza la entrega de los paquetes de información).

14 Para conocer la dirección IP de un servidor podemos utilizar la utilidad de línea de comandos ping direccion: p.ej. *ping www.google.es*

Una URL se expresa según el guión-tipo siguiente:

```
esquema://host.dominio:puerto/rutaRecurso
```

Por ejemplo http://openlayers.org:80/blog/

El cliente, generalmente a través de un navegador web, se conecta al servidor y le solicita/envía un recurso. En cada servidor existe un software específico, el 'servidor web', mayoritariamente Apache o Microsoft Internet Information Server (IIS), que recibe la petición y la atiende devolviendo/recibiendo el recurso y presentando un código de operación correcta o error. Cada petición es independiente de la anterior de cara al servidor (es un protocolo sin estado), aunque hay mecanismos como las *cookies* que permiten ligar varias peticiones dentro una misma 'sesión' de trabajo.

El ciclo de carga típico de una página web con HTTP es:

1. Se introduce una dirección en el navegador web (p.ej. http://www.google.es).
2. El nombre simbólico debe ser traducido en una dirección IP para establecer la conexión efectiva. Esto se consigue gracias al protocolo de resolución de nombres (DNS). Nuestro equipo consulta a ese servicio de manera transparente y obtiene que http://www.google.es equivale p.ej. a la dirección IP 64.249.92.104
3. El navegador web de nuestro equipo realiza a continuación una petición HTTP a la dirección IP resuelta. Para ello establece una conexión TCP con el servidor remoto por el puerto 80. Éste servidor web tiene un programa que está constantemente a la espera de peticiones.
4. El servidor web recibe la petición y la atiende, devolviendo una respuesta a través de la conexión TCP.
5. El cliente recibe el resultado y éste se canaliza hasta el programa navegador. Cuando el documento HTML incorpora recursos múltiples (p.ej. imágenes, ficheros de Javascript, hojas de estilo...), se realizan varias peticiones HTTP, una por recurso y su resultado se ensambla en el navegador, el cual interpreta el contenido y lo dibuja / dota de comportamiento: posicionamiento de elementos, fuentes de texto y colores, animaciones, etc.

Todos los aspectos previos de este apartado nos permiten entender algunos puntos importantes a tener en cuenta al trabajar con OpenLayers:

- Los servidores web procesan peticiones HTTP y los *servidores de mapa* se apoyan en ellos para hacer llegar al navegador recursos,

generalmente imágenes (de tipo gif, jpg, png, etc.), con la información geográfica de nuestro interés. Cuando en OpenLayers usemos p.ej. una capa de *GoogleMaps,* nuestro navegador pide al servidor teselas en formato de imagen de 256 x 256 píxeles, con peticiones HTTP de recursos como:

http://mt0.google.com/vt/lyrs=h@140&hl=es&src=api&x=1000&y=749&z=11&s=Galil

Ilustración 7: Tesela de mapa de GoogleMaps en formato png

En otras ocasiones el recurso solicitado es vectorial, como cuando solicitamos conectarnos a un servidor WFS y lo que viaja desde el servidor es información en un formato de texto como GML (XML).

- Para poder acceder a ciertos servicios de información geográfica desde nuestras páginas habremos de conocer la *dirección IP* del servidor o su nombre registrado y la *URL del recurso* que expone. P.ej. para consumir un servicio WMS del PNOAA en nuestro visor debemos conocer previamente su URL: http://www.idee.es/wms/PNOA/PNOA

- Para compartir el trabajo realizado, ya sea un geomashup o una página web estándar, habremos de contar con un servidor web para trabajar con HTTP y unos clientes con navegador web, todos ellos con sus propiedades de red bien configuradas (dirección IP, puerta de enlace y servidor DNS).

- Si al trabajar con OpenLayers hubiera problemas de conectividad entre nuestra web y los servicios de los que depende (p.ej. un servicio WFS externo del que no obtenemos respuesta), es recomendable monitorizar el tráfico de red. Si nos centramos en HTTP, son muy

útiles *Fiddler* y la extensión para Firefox llamada Firebug. Para labores más complejas y todo tipo de protocolos, tenemos un analizador de protocolos o '*sniffer*' como Ethereal-WireShark.

2.2. Los lenguajes para la web

Los próximos apartados son una breve introducción a los lenguajes más importantes para el desarrollo web con OpenLayers: HTML, CSS y Javascript. Al final de cada apartado se incluyen referencias para profundizar en el aprendizaje.

2.2.1. Diseñando la estructura de una página con HTML

HTML (HyperText Markup Language) es el lenguaje utilizado para construir páginas web. Sus comandos se incluyen en documentos que nuestros equipos descargan del servidor y son interpretados por el navegador (Mozilla Firefox, Google Chrome, Opera, Internet Explorer...).

HTML es un lenguaje de marcado: sus comandos son marcas o 'etiquetas' que describen un elemento, escrito en texto común. La sintaxis es `<etiqueta>elemento</etiqueta>` y las etiquetas pueden ir anidándose. Cada bloque etiquetado puede llevar un identificador único en el documento, su id: `<etiqueta id="codigo01">elemento</etiqueta>`

La estructura general de un documento HTML es:

```
<html>
	<head>
		<!-- Titulo, código Javascript, estilos -->
		<title>Título de la página</title>
		<script></script>
		<link/>
	</head>
	<body>
		<!-- Contenido del documento -->
	</body>
</html>
```

Los ficheros son de texto plano (tradicionalmente con la extensión *.html*) y por ello pueden ser examinados o editados con un simple editor de texto como el bloc de notas. A continuación incluiremos las etiquetas más habituales e importantes para nosotros que pueden aparecer dentro de un documento HTML.

Etiquetas generales

<HTML>

<HEAD>

<BODY>

Etiquetas de la cabecera (HEAD)

<TITLE> Título de la página

```
<TITLE>Mi primera página con OpenLayers</TITLE>
```

<META> Para almacenar metadatos del documento. Quién lo creó, cuándo, palabras clave... Ejemplo:

```
<META name=”Keywords” content=”SIG, openlayers, visor”>
```

<SCRIPT> Para recoger código de cliente en Javascript, bien incluido en el propio documento o bien en un fichero .js externo. Veremos la sintaxis básica del lenguaje Javascript más adelante. Ejemplos:

Con código en línea:

```
<SCRIPT type=”text/javascript”>
      alert('Hola mundo');
</SCRIPT>
```

Referenciando un fichero externo de javascript:

```
<SCRIPT src=”http://www.openlayers.org/api/OpenLayers.js”
type=”text/javascript”></SCRIPT>
```

<STYLE> y **<LINK>** Para incluir código CSS con estilos, bien en línea o bien apuntando a un fichero .css externo. Ejemplos:

En línea:

```
<STYLE type=”text/css”>
      body{ font-color: Red; font-size: 14px; }
</STYLE>
```

Fichero externo:

```
<link rel="stylesheet" href="estilo.css" type="text/css" />
```

Etiquetas del cuerpo (BODY)

<DIV> Etiqueta de bloque o divisor, para organizar secciones. Ejemplo:

```
<DIV id=”mapa”>Aquí la sección de mapa de OpenLayers<DIV/>
```

<P> Etiqueta de párrafo. Ejemplo:

```
<P>Contenido de un párrafo: LOREM IPSUM...<P/>
```

**
** Salto de línea.

<HR/> Línea horizontal separadora.

<H1> hasta **<H6>** Cabeceras o rótulos jerarquizados.

<A> Etiqueta para hiperenlaces.

```
<A href=”http://www.openlayers.org”>Enlace a la web OL<A/>
```

<IMG> Imágenes accesibles desde la página.

```
<IMG src="http://www.google.es/logos/logo.gif"/>
```

<TABLE> Tabla estándar, con etiquetas de tipo fila (tr), cabecera (th) y datos (td)

```
<TABLE border=”1px black solid”>
      <TR><TH>Columna A</TH><TH>Columna B</TH></TR>
      <TR><TD>Fila 1-Col.A</TD><TD>Fila 1-Col.B</TD></TR>
      <TR><TD>Fila 2-Col.A</TD><TD>Fila 2-Col.B</TD></TR>
</TABLE>
```

<LI>, **<OL> y <UL>** OL para listas ordenadas (con numeración) y no ordenadas (con viñetas).

```
<OL>
      <LI> Elemento primero (con numeración)</LI>
      <LI> Elemento segundo </LI>
</OL>
<UL>
      <LI> Elemento primero (con viñetas) </LI>
      <LI> Elemento segundo </LI>
</UL>
```

<FORM> y etiquetas de controles de entrada. Para crear un formulario y recoger información del usuario. Los controles más habituales son las

cajas de texto (textboxes), casillas de verificación (checkboxes), radiobotones (radio buttons) o las listas desplegables (select). Ejemplo:

```
<FORM action="http://unsitio.com/registro" method="post">
      <LABEL for="nombre">Nombre:</LABEL>
            <INPUT type="text" id="nombre"><BR/>
      <LABEL for="apellido">Apellidos:</LABEL>
            <INPUT type="text" id="apellido"><BR/>
      <INPUT type="radio" name="sexo" value="Varón">Varón<BR/>
      <INPUT type="radio" name="sexo" value="Mujer">Mujer<BR/>
      <INPUT type="checkbox" name="reserva" value="Reserva">
      <SELECT name="tipo-abono">
            <OPTION>Semanal</OPTION>
            <OPTION>Mensual</OPTION>
            <OPTION>Anual</OPTION>
      </SELECT>
      <INPUT type="submit" value="Enviar">
      <INPUT type="reset">
</FORM>
```

Aunque con ciertas restricciones, los controles input pueden utilizarse fuera de un formulario. Alternativamente al input de tipo botón se puede utilizar también la etiqueta `<button type="button">Clic aquí</button>`

Para insertar un comentario informativo, que no será procesado por el navegador se utilizan las etiquetas: `<!-- con el texto dentro -->`

A pesar de que HTML incluye otras etiquetas para reflejar la apariencia (p.ej. <B> para mostrar texto negrita), es un lenguaje que debe utilizarse para recoger el contenido y la estructura de la web, pero no su apariencia. Para disposición de los elementos, colores, bordes, fuentes, es preferible usar el lenguaje CSS.

Para reflejar el uso de una versión concreta de HTML es recomendable incluir en su inicio la declaración del DOCTYPE. Para el caso de HTML 4 estricto, utilizar la siguiente línea antes del primer tag <html>:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Referencias complementarias

- Existen más etiquetas de HTML, cada una con sus propiedades específicas. Actualmente, la mayoría de las páginas web están escritas conforme al estándar HTML 4.01, si bien está bastante avanzada la especificación HTML5. El listado completo de las etiquetas y sus propiedades para ambas versiones se puede consultar en http://www.w3schools.com/tags/

- En esa misma página existe una **sección interactiva** muy práctica para estudiar cada tipo de etiqueta (*Try it yourself*), p.ej. en el apartado de tablas:

  http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_table_test

2.2.2. Dando estilo con hojas en cascada (CSS)

CSS (*Cascading Style Sheets*) establece de una manera estructurada las pautas visuales con las que se muestran los contenidos dentro de un documento HTML: posiciones de bloques, tamaños, colores, márgenes, etc..

Se basa en la creación de **reglas**, que vinculan unas declaraciones de estilo con una etiqueta/s mediante un selector. Una declaración consta de una propiedad y un valor compatible asignado. P.ej. para conseguir un tamaño de 22px y color rojo para un encabezado de nivel 1 utilizaríamos la regla:

```
h1 { color: Red; font-size: 22px; }
```

donde h1 es el selector y hay dos declaraciones de estilo.

Las reglas pueden aparecer de varias maneras:

1. Directamente aplicadas sobre una etiqueta en el HTML, mediante la propiedad style. No necesita selector ni llaves. P.ej.:

   ```
   <p style="text-align: right; font-size:10px"> Párrafo alineado a la derecha </p>
   ```

2. En una sección de estilos del documento HTML dentro de la cabecera, con el tag <style>. En este caso, el estilo puede aplicarse sobre varias etiquetas de 1 página.

   ```
   <html>
       <head>
           <style type="text/css">
               /* Esto es un comentario en CSS */
               div { padding-left: 5px }
               p { font-size: 10px }
   ```

```
            </style>
        </head>
        <body>
            <div><h1>Cabecera</h1></div>
            <p id=”parrafo01”>Lorem ipsum</p>
            <p id=”parrafo02”>Lorem ipsum</p>
        </body>
</html>
```

3. En un fichero separado, con extensión *css*. Así el estilo se puede aplicar a varias etiquetas de varias páginas. El contenido del fichero estilo.css sería una sucesión de reglas, como:

```
div { padding-left: 5px }
p { font-size: 10px }
```

y su uso desde una página web:

```
<html>
    <head>
        <link rel="stylesheet" type="text/css"
href="estilo.css">
    </head>
    <body><!-- contenidos con estilo --></body>
</html>
```

Los selectores CSS son muy variados. Los 3 más importantes son:

- *Selector de elemento HTML*. Los ejemplos que hemos visto previamente corresponden a esa categoría. El estilo se aplica a todas las etiquetas html de ese tipo bajo el ámbito de influencia de la regla.

```
a { font-family: Arial, Verdana; }
```

- *Selector por identificador.* Se usa el atributo id de una etiqueta para seleccionarla, utilizando como prefijo la almohadilla. P.ej.:

```
<p id=”parrafo01”></p>
```

se le puede aplicar un estilo mediante una regla como:

```
#parrafo01 { background-color: black }
```

- *Selector por clase.* A diferencia del id, una clase puede ser compartida por varias etiquetas html. Y una etiqueta puede tener varias clases asociadas. El selector por clase va precedido por punto. P.ej.:

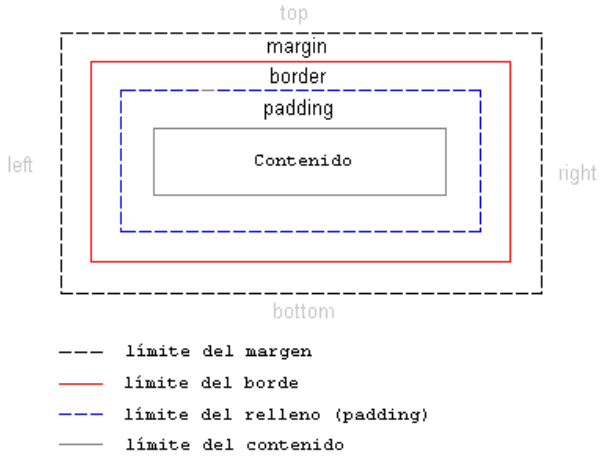
  ```
  <p class="citaLibro"></p>
  ```

  se le puede aplicar un estilo mediante una regla como:

  ```
  .citaLibro { font-style: italic }
  ```

Si nos referimos a aspectos del diseño como el tamaño de bloques, los márgenes, bordes, relleno... es importante conocer las propiedades de lo que se denomina el "modelo de cajas de CSS":

Ilustración 8: Modelo de cajas CSS



Fuente: Tutorial de CSS de HTML.NET, disponible en http://es.html.net/tutorials/css/lesson9.php

Siguiendo este modelo, cualquier elemento de una página web (un div, un p, un span...) está dentro de una "caja rectangular". Ajustando las propiedades de esa caja, mostradas en la figura previa, se le puede dar:

- unas dimensiones mediante la expresión de un ancho y alto predefinidos, con las propiedades *width* y *height*:

  ```
  <div style="width: 200px; height: 200px"></div>
  ```

- un borde visible, mediante la propiedad *border*:

  ```
  <div style="border: 1px red solid"></div>
  ```

- un margen exterior transparente, más allá del borde, mediante la propiedad *margin*:

```
<div style="margin-top: 20px; margin-left: 30px"></div>
```

- un margen interior transparente, entre el borde y el contenido, mediante la propiedad *padding*:

```
<div style="padding: 40px"></div>
```

Las cajas y otras propiedades de estilo controlan otras propiedades más avanzadas de CSS, tales como el posicionamiento relativo de elementos o las capas flotantes (position, float).

Referencias complementarias

- Para un listado más exhaustivo de selectores, puede visitarse p.ej.: http://www.anieto2k.com/2006/09/06/selectores-css-que-deberias-conocer
- Varios ejemplos de estilos CSS comunes pueden encontrarse en: http://www.w3schools.com/css/css_examples.asp
- Para aspectos más avanzados de CSS como el posicionamiento: http://www.w3schools.com/css/css_positioning.asp
- Una serie de documentos muy completa y en español con todos los aspectos detallados sobre CSS puede descargarse en: http://www.librosweb.es/css/index.html

2.2.3. Funciones y eventos con Javascript

HTML + CSS nos proporcionan unas capacidades importantes para recoger contenidos y representarlos de acuerdo al estilo que consideremos más adecuado. Pero por sí sólos estos dos lenguajes sólo pueden presentar contenidos estáticos. Para dotarlos de interactividad y mayor riqueza es necesario utilizar lenguajes de programación que nos permitan cambiar dinámicamente los contenidos y los estilos. Este papel es cubierto en gran parte por Javascript.

Javascript es un lenguaje de scripting que se ejecuta *dentro del propio navegador web* (por eso se denomina **lenguaje en cliente**) y lo hace de manera interpretada, es decir sin tener que ser compilado. Tras un periodo inicial en el que solo se usaba para lograr efectos visuales vistosos, hoy en día constituye una pieza angular de la Web 2.0. De hecho está detrás de la

mayoría de *mashups* y especialmente de las librerías de mapa[15]. OpenLayers es una librería hecha en Javascript y debe utilizarse con este lenguaje. Cuanto mejor se conozca Javascript (JS), mejores proyectos se podrán hacer con esta librería.

Al igual que con CSS, el código escrito en JS puede ubicarse en varios lugares:

1. En una etiqueta en el HTML. En este caso, el código se comporta como un *manejador de eventos* y la sintaxis tipo es:

   ```
   <etiqueta evento=”codigo_Javascript”></etiqueta>
   ```

   Cada etiqueta tiene sus eventos permitidos, habitualmente acciones desencadenadas por el usuario (como hacer clic, pasar el ratón por encima o elegir un texto). P.ej. para vincular una acción al clic de un botón realizaríamos:

   ```
   <button onClick=”alert('holaMundo');”></etiqueta>
   ```

   De esta manera se dota de interactividad al HTML. Cuando queramos lograrlo, habremos de conocer los eventos a los que responde el control y asignar una función de Javascript.

2. Dentro de una etiqueta de <script>, preferiblemente en la cabecera del documento HTML y organizado en funciones. P.ej:

   ```
   <html>
         <head>
               <script type=”text/javascript”>
               alert('Cargando...');
               // Esto es un comentario en Javascript
               function Saludo(){
                     var n = document.getElementById('name').value;
                     alert('Hola ' + n);
               }
               </script>
         </head>
   ```

15 En este rol tiene mucho que ver una técnica conocida como AJAX (Asynchronus Javascript And XML), que consigue realizar peticiones de datos a los servidores web sin realizar una recarga completa de la página, y dota con ello de gran dinamismo a las páginas web, asemejándolas en este sentido a aplicaciones de escritorio 'tradicionales'. Ajax está detrás de proyectos como el API de GoogleMaps y se usa en la mayoría de webs modernas.

```
        <body>
            <input type=”text” id=”name”></input>
            <button onClick=”Saludo();”></button>
        </body>
</html>
```

Conforme el documento se carga, el navegador revisa el script. Si hay código invocando una función, como el '*alert*', éste se ejecuta. Si encuentra una definición de función (como *Saludo*) se carga y queda disponible para usos posteriores.

3. En un fichero externo, con extensión js. Para situaciones en las que las funciones quieren reutilizase en varias páginas web o simplemente conseguir una mejor organización del código. El contenido del fichero en Javascript sería:

```
function Saludo(){
    var n = document.getElementById('nombre');
    alert('Hola ' + n);
}
```

y su uso desde una página web:

```
<html>
    <head>
        <script type="text/javascript" src="fichero.js">
    </head>
    <body><!-- uso de la función --></body>
</html>
```

A continuación repasaremos algunos de los componentes básicos del lenguaje que nos serán útiles al trabajar con OpenLayers:

Sentencias básicas

```
var tituloMapa = 'Mapa inicial'; //declaración de una variable
alert(tituloMapa);      //invocar una función con un parámetro
var nuevaCoordenada = coordX + 100; // operar con dos valores
```

Tipos de variables

```
// TEXTO
var nombreCapa = 'Servicio PNOA'; // comillas simples o dobles


// NUMEROS (enteros, decimales...)
var numeroDeCapas = 5;
var latitud = 45.48;


// BOOLEANOS: cierto o falso
var tieneRegistros = true;
var laCapaEsVisible = false;


// NULO e indefinido
var mapa = null;
var otroMapa; // undefined


// ARRAYS: listas de elementos ordenados
var ciudades = ['Santander', 'Madrid', 'Burgos']; // otra lista
var ciudad = ciudades[2]; // ciudad vale 'Burgos'
ciudades.push('Torrelavega'); // añadir un elemento al final
/* Métodos disponibles para arrays explicados en
http://www.w3schools.com/jsref/jsref_obj_array.asp */
// OBJETOS
var unaPosicion = new Object();
unaPosicion.nombre = “Pueblo”;
unaPosicion.coordenadas = [-3.44, 45.88];
// formato JSON[16]
var unaCasa = {
      color: 'Blanco',
      numeroHabitaciones: 4,
```

16 JSON: el formato JavaScript Object Notation es muy utilizado en los servicios web actuales y cada vez se extiende más, frente a otros formatos más 'pesados' como xml.

```
        habitantes: [
                {nombre: 'John Doe', edad: 33},
                {nombre: 'Jack Bauer', edad: 41}
        ]
};
var john = unaCasa.habitantes[0];
var edad = john.edad; // vale 33
```

Operadores

```
// Concatenar cadenas
alert('Hola ' + 'mapa'); // muestra 'Hola mapa'
// Operadores aritméticos
var resultado = (1 + 2 – 4) * 5 / -1     // vale 5
// Comparación y operadores lógicos
var numero = 4;
(numero === 4) // igual
(numero !== 5) // distinto
(numero > 2) // mayor que
(numero >= 4) // mayor/igual que
(numero < 5) // menor que
(numero <= 4) // menor/igual que
var condicion01 = true;
var condicion02 = false;
var A = (condicion01 && condicion02) // AND: A es falso
var B = (condicion01 || condicion02) // OR: B es cierto
var C = !(condicion02) // NOT: C es cierto
```

Código condicional

```
// IF... ELSE - Decidir qué ejecutar en base a una condición
var javascriptEsImposibleDeAprender = false;
if (javascriptEsImposibleDeAprender){
        var quien = 'yo';
```

```
        tirarLaToalla(quien);
}else{
        practicarMasYAvanzar();
}

// SWITCH - Decidir qué ejecutar en base a un valor
var opcionElegida = 3;
switch (opcionElegida) {
        case 1:
                alert('La primera');
                break;
        case 2:
                alert('La segunda');
                break;
        case 3:
                alert('La tercera'); // ésta sí se ejecuta.
                break;
        default:
                alert('NO has elegido 1, 2 o 3');
}
```

Código iterativo

```
// FOR - Realizar una acción varias veces
var indice;
for (indice=0; i<=10; i++){
        alert(“Ahora i vale: “ + i);
}
// la función alert se ejecutará 11 veces

// WHILE - Realizar una acción mientras se cumpla una condición
var cantidad = 100;
while(cantidad < 1000){
```

```
        cantidad = cantidad + 100;
    }
```

Además de las estructuras de código que hemos visto, con Javascript se puede interactuar con el documento html mediante lo que se conoce como *el modelo DOM*, una abstracción que permite ver el documento html como un árbol con nodos manipulables. Esto permite utilizar funciones de búsqueda para obtener y manipular elementos como:

```
var listaDivs = document.getElementsByTagName('div');
var unEnlace = document.getElementById('enlace05');
```

Con Javascript se puede acceder a varios objetos y funciones predefinidos del navegador. De ellos, los más destacados son los pertenecientes al objeto window, el cual permite funciones como:

```
window.alert('un mensaje');
window.document.write("<br>");
```

Si queremos que el código para acceder al DOM y otras propiedades del navegador sea más flexible y sencillo, es recomendable utilizar librerías de Javascript de apoyo como **jQuery**.

Referencias complementarias

- Se puede encontrar mucha información sobre Javascript en la web. Es interesante el sitio http://eloquentjavascript.net que tiene elementos interactivos para practicar con los elementos más básicos del lenguaje.
- Como la programación en Javascript es tan usada actualmente se han programado librerías gratuitas muy útiles, que facilitan su uso y permiten aislar al programador de las diferencias entre navegadores. Contienen funciones como: efectos visuales, generación dinámica de html, selectores complejos, comunicación vía AJAX con servicios externos, etc. Las más utilizadas actualmente son probablemente jQuery www.jquery.com y Mootools: http://mootools.net/

2.2.4. El papel de los lenguajes de servidor

Los lenguajes de servidor son aquellos cuya ejecución se produce en el servidor web, en lugar de ejecutarse dentro del navegador web como lo hace Javascript. Su objetivo es salvar las limitaciones del entorno de ejecución del cliente, principalmente dotando al programador de una mayor capacidad para acceder a recursos. Aunque en esta asignatura no se trabaja con lenguajes de servidor, es importante al menos presentarlos y tener unas referencias por si deseamos profundizar en su conocimiento.

El servidor web, una vez recibe la petición HTTP del cliente, tiene libertad para utilizar los medios precisos para responder: librerías de cálculo, lectura de ficheros o bases de datos, peticiones a nuevos servidores, etc. El único requisito es que los resultados se traduzcan antes de ser enviados a un lenguaje reconocible desde el navegador (habitualmente HTML, XML o JSON).

Las tecnologías en servidor son muy variadas:

- **CGI** (Common Gateway Interface). El sistema más antiguo, que permite la ejecución en el servidor de programas hechos en C++, Visual Basic, Perl, Python... Un servidor de mapas muy extendido como es Mapserver trabaja con esta tecnología.
- **Perl**. Un lenguaje interpretado, con muchas funciones para el procesamiento de texto.
- **PHP**. Libre y gratuito, muy extendido en páginas de foros y comunidades sociales. Hay un un software especializado en gestión de contenidos, llamado Drupal, que está hecho con PHP, y cuenta con un módulo que integra funciones geográficas con OpenLayers: http://drupal.org/project/openlayers
- **Python**. Un lenguaje de script multipropósito, utilizado en ámbitos muy diversos (computación científica, geoprocesamiento, administración de sistemas...).
- **ASP.NET**. La alternativa web en servidor de Microsoft, dentro de su estrategia de herramientas y servicios .NET, con varios lenguajes disponibles: VisualBasic .NET y sobre todo C#.
- **JSP**. La tecnología de páginas dinámicas de Java, dentro de la arquitectura general J2EE.

En la actualidad, la tendencia generalizada es que las corporaciones realicen sus aplicaciones web en uno o varios lenguajes de servidor (C#, Java, Python...), y que internamente muchas estén organizadas en

servicios. Por decisión de los gestores algunos de dichos servicios se pueden abrir al uso desde clientes externos, bajo pago o gratuitamente, y generalmente adoptando estándares como los servicios web vía SOAP y los servicios web tipo REST (que se traducen generalmente en flujos de datos XML y JSON, respectivamente).

Referencias complementarias

- Como lenguaje para iniciarse en la programación web en el servidor, recomendamos PHP o Python. Para PHP tenemos cursos online como http://www.programacionweb.net/cursos/curso.php?num=10 y para PYTHON recomendamos buscar en google por marcos de desarrollo como *Pylons* o *Django.*
- Si en nuestro ámbito de referencia ya se trabaja con .NET o J2EE, entonces la alternativa clara es trabajar respectivamente en ASP.NET y JSP. Se puede encontrar buenos recursos en http://www.asp.net/ y http://www.jsptube.com/, respectivamente.

3. Aprendiendo a desarrollar con OpenLayers

Como base para la realización de los ejercicios que se entregan en el aula, en este apartado presentamos las capacidades generales de OpenLayers y también un listado de recursos de aprendizaje.

En el primer apartado nos centraremos en los aspectos más básicos de la librería: cómo referenciar la librería para crear un mapa, conocer los distintos tipos de capas y los controles básicos de interacción.

En el segundo bloque presentaremos un conjunto de recursos interesantes para el lector: dónde encontrar ejemplos, manuales y cómo usar la documentación del API, para que éste pueda avanzar en el aprendizaje de manera autónoma.

3.1. ¿Cómo se usa OpenLayers?

OpenLayers es una librería de mapas hecha totalmente en Javascript, con tecnología AJAX y sin dependencias en el servidor, que toma como base de desarrollo otras dos librerías de JS como son *Rico* y *Prototype*. Su código fuente es libre y toda la documentación relevante puede ser encontrada en su página web oficial.

Ilustración 9: Sitio web oficial de OpenLayers

www.openlayers.org

La última versión de librería se encuentra en la siguiente URL: http://www.openlayers.org/api/OpenLayers.js

Para utilizarla, basta con apuntar a esa dirección en una página web de la siguiente manera:

```
<script type="text/javascript"
src="http://www.openlayers.org/api/OpenLayers.js">

</script>
```

Utilizando ese fichero .js, en la carpeta **api**, estaremos seleccionando la última versión estable de la librería. Si cambiamos la carpeta *api* por **dev** apuntaremos a la versión en desarrollo (dev-elopment), la cual nos permitirá probar las nuevas funcionalidades en marcha.

Si lo deseamos podemos descargar una versión concreta de la librería y desplegarla en nuestro servidor web. Aquí están todas las que se han liberado hasta el momento: http://www.openlayers.org/download/

3.1.1. Creación de un mapa básico

El objetivo fundamental de la librería es poder publicar un mapa dentro de una página web, lo cual se puede hacer de manera rápida y sencilla con el siguiente código:

```
<html>

<head>

      <script type="text/javascript"
      src="http://www.openlayers.org/api/OpenLayers.js">

      </script>

      <script type="text/javascript">

      var mapa;

      function holaMapa(){

            mapa = new OpenLayers.Map('capaMapa');

            var capaWMS = new OpenLayers.Layer.WMS(

                  "OpenLayers WMS",

                  "http://vmap0.tiles.osgeo.org/wms/vmap0",

                  {layers: "basic"});

            mapa.addLayer(capaWMS);

            mapa.zoomToMaxExtent();

      }

      </script>

</head>
```

```
<body onload="holaMapa()">

      <div id='capaMapa' style="top: 0px; width: 700px; height:
      400px;">

      </div>

</body>

</html>
```

El resultado de ejecutar la página puede verse en la siguiente ilustración:

Ilustración 10: Hola mundo con OpenLayers

En el código puede verse una constante sobre el estilo de programación con la librería y es que está orientada a clases, las cuales permiten instanciar los distintos objetos (mapas, capas, controles...).

P.ej. con la línea `mapa = new OpenLayers.Map('capaMapa');` se crea un objeto de la clase *OpenLayers.Map* mediante su función constructora, al cual se le hace llegar un parámetro, que es el nombre del div donde se pintará el mapa.

Con la sentencia `var capaWMS = new OpenLayers.Layer.WMS("OpenLayers WMS", "http://vmap0.tiles.osgeo.org/wms/vmap0", {layers: "basic"});` se hace lo mismo con la función constructora de la capa WMS, a la cual se suministran 3 parámetros, siendo el último un objeto de tipo *params = {clave: valor}*

3.1.2. ¿Qué tipos de dato puedo cargar con OpenLayers?

Openlayers trabaja, a diferencia de otros APIs, con un amplio número de orígenes de datos (servicios de mapa comerciales, servidores de mapa

opensource, servidores propietarios...), y además lo hace siempre que sea posible de la mano de los estándares OGC más extendidos (WMS, WFS, SLD...) por lo que proporciona mucha flexibilidad al programador. Es una 'navaja suiza' para consumir servicios de mapa, que separa los datos de las herramientas.

En OpenLayers existe el concepto de *capa base*, que es la capa principal de fondo activa, de la cual se toma el sistema de proyección y los niveles de zoom. Sobre ésta pueden añadirse más capas simultáneas, a modo de *overlay*, sean raster o vectoriales. Existen muchos tipos de capas:

Ilustración 11: Tipos comunes de capas usados en OpenLayers

Capa	Descripción
Google	Raster. EPSG 900913[17]. Varias opciones: satélite, mapa carreteras, híbrido y relieve.
Bing	Raster. EPSG 900913. Opciones: satélite, mapa carreteras e híbrido. Nota: hasta la v2.10 sólo está disponible el tipo de capa VirtualEarth (sin acceso directo a la caché de mapas de Microsoft).
WMS	Raster. EPSG variable. Permite seleccionar las capas del WMS.
OpenStreetMap	Raster. EPSG 900913.
GML	Vectorial. EPSG variable.
Vector	Vectorial. EPSG variable. Permite cargar formatos como GeoJSON, KML...

Las capas se localizan en el espacio de nombres **OpenLayers.Layer**. La capa *Google* cuenta p.ej. con la clase correspondiente *OpenLayers.Layer.Google.*

Todas las capas tienen la misma manera de crearse y añadirse al mapa:

1. usar el constructor específico de capa, con los parámetros requeridos.

```
var capaR = new OpenLayers.Layer.Google("Carreteras", {type:
        google.maps.MapTypeId.ROADMAP, numZoomLevels: 20} );

var capaH = new OpenLayers.Layer.Google("Hibrido", {type:
        google.maps.MapTypeId.HYBRID, numZoomLevels: 20});

var aerea = new OpenLayers.Layer.Bing({key: apiKey, type: "Aerial"});
```

17 Proyección 'Mercator esférica'. Equivalente al EPSG: 3587

```
var WMS = new OpenLayers.Layer.WMS("WMS",
        "http://vmap0.tiles.osgeo.org/wms/vmap0", {layers: 'basic'});

var kml = new OpenLayers.Layer.GML("Pueblos", "datos/pueblos.kml",
        {format: OpenLayers.Format.KML});
```

2. añadir la capa al mapa, individualmente o con otras más a la vez:

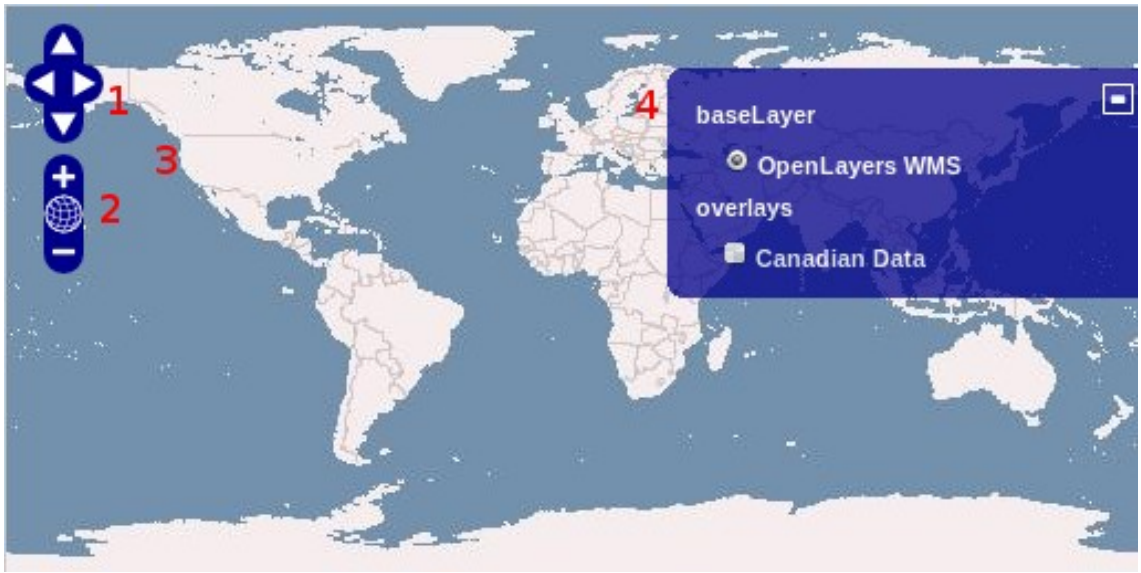
```
mapa.addLayer(capaR);

mapa.addLayers([aerea, WMS]);
```

Además de los tipos mostrados existen otros orígenes de datos admitidos (ver apartado correspondiente del API). Todas ellas heredan de *OpenLayers.Layer* y poseen ciertas propiedades y métodos comunes (extensión, resolución, sistema de coordenadas, unidades, visibilidad...): http://dev.openlayers.org/apidocs/files/OpenLayers/Layer-js.html

3.1.3. Controles básicos para interactuar con OpenLayers

Los controles son componentes que permiten la interacción del usuario con el mapa de OpenLayers. Algunos se muestran en pantalla como el control de capas (*LayerSwitcher*) y otros no son visibles y controlan ciertos aspectos de la interacción, como p.ej el control de selección para entidades vectoriales (*SelectFeature*).

Ilustración 12: Controles visuales comunes en OpenLayers (1 de 2)



	Control	Descripción
1	**PanPanel**	Flechas para el desplazamiento en los cuatro puntos

		cardinales.
2	**ZoomPanel**	Incluye en un panel 3 botones: más zoom más, menos zoom y zoom a la extensión. Cada botón se corresponde con un control utilizable separadamente.
3	**PanZoom**	Combina los controles 1 y 2
4	**LayerSwitcher**	Controla la capa base activa si hay varias y permite cambiar la visibilidad de los *overlays*

Ilustración 13: Controles visuales comunes en OpenLayers (2 de 2)

	Control	Descripción
1	**PanZoomBar**	Incluye PanPanel + ZoomBar.
2	**ScaleLine**	Línea de escala actual.
3	**OverviewMap**	Mapa de situación.
4	**Permalink**	Generar un enlace al encuadre actual de mapa.
5	**MousePosition**	Coordenadas actuales del cursor sobre el mapa.

Los controles se localizan en el espacio de nombres **OpenLayers.Controls**. Así por ejemplo el control *LayerSwitcher* se corresponde con la clase *OpenLayers.Controls.LayerSwitcher.*

En la configuración simple de mapa de OpenLayers, se agregan por defecto las funciones de desplazamiento y zoom (control *PanZoom),* pero si queremos manejar más controles, los podemos indicar en el constructor del mapa o agregarlos individualmente.

Por ejemplo:

```
var opciones = { controls: [
        new OpenLayers.Control.Navigation(),
        new OpenLayers.Control.KeyboardDefaults(),
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.Scale()
    ]};
mapa = new OpenLayers.Map('map', opciones);
var autoria = new OpenLayers.Control.Attribution();
mapa.addControl(autoria);
```

Una vez vinculado un control al mapa, el primero lo referencia en su propiedad '*map*' y el segundo lo incluye en su colección '*controls*'. Para ver todos los controles y sus propiedades:

http://dev.openlayers.org/apidocs/files/OpenLayers/Control-js.html

3.2. Recursos para profundizar en el aprendizaje de OpenLayers

OpenLayers está en constante mejora y ampliación y la mejor manera de continuar el aprendizaje es con los siguientes recursos:

Ilustración 14: Guía rápida de recursos para OpenLayers

	Recurso	Dirección
1º	Página oficial	**http://www.openlayers.org**
2º	Ejemplos oficiales	**Versión concreta:** **http://dev.openlayers.org/releases/OpenLayers-2.10/examples/** **Desarrollo:** **http://openlayers.org/dev/examples/**

3º	API de OpenLayers	**Versión concreta:** **http://dev.openlayers.org/releases/OpenLayers-2.10/doc/apidocs/files/OpenLayers-js.html** **Desarrollo:** **http://dev.openlayers.org/apidocs/files/OpenLayers-js.html**
4º	Manual de OpenLayers en español – Descartes	**http://www.ingemoral.es/pages3/manualOpenLayers.html**
5º	Taller de trabajo con OpenLayers - OpenGeo	**http://workshops.opengeo.org/openlayers-intro/**
6º	Ejemplos - Geotribu	**http://geotribu.net/taxonomy/term/15**
7º	OpenLayers & OpenStreetMap	**http://wiki.openstreetmap.org/wiki/OpenLayers**
8º	Lista de correo OpenLayers - Usuarios	**http://openlayers.org/mailman/listinfo/users**
9º	Lista de correo OpenLayers - Programadores	**http://openlayers.org/mailman/listinfo/dev**
10º	Lista de correo – Comunidad hispana de osGeo	**http://www.osgeo.org/content/faq/mailing_lists_ES.html**
11º	Cursos tecnologías espaciales con software libre – Comunidad hispana de osGeo	**http://wiki.osgeo.org/wiki/Category:Cursos**

Una buena manera de utilizar estos recursos podría ser:

(a) Surge una duda con OpenLayers y nos dirigimos al navegador web.

(b) Buscamos directamente en Google, con unas buenas palabras clave y operadores, lo cual nos puede revelar recursos directamente aplicables.

(c) En caso de no encontrar la solución usamos los ejemplos de OpenLayers (recurso 2º), refinando la búsqueda con el filtro integrado. Otras vías de ejemplo interesantes son los recursos 4º, 5º y 6º.

(d) Los ejemplos explican gran parte de la librería, pero a veces no es suficiente. En ese caso,nos dirigiremos al API (recurso 3º), que documenta de manera detallada cada clase. Generalmente navegaremos comenzando con un objeto principal relevante (por ejemplo qué parámetros tiene un constructor de capa WFS > *OpenLayers.Layer.WFS* o cómo generar un encuadre > *OpenLayers.Bounds*) y examinando sus métodos y sus propiedades.

(e) Normalmente en el nivel previo se resuelven las dudas más habituales. Si no es así, utilizar las listas de correo (recursos 10º, 8º y 9º). Una vez en ellas es recomendable utilizar previamente las herramientas de búsqueda, por si ya se hubiera formulado una consulta similar y si no es así formular la consulta de manera concisa y clara.