

LINEAMIENTOS PARA EL DESARROLLO DE APLICACIONES SIG WEB

ANDREA VIVIANA YANZA HURTADO

**Universidad EAFIT
Escuela de Ingeniería
2013**

LINEAMIENTOS PARA EL DESARROLLO DE APLICACIONES SIG WEB

ANDREA VIVIANA YANZA HURTADO

**Trabajo de grado presentado como requisito para optar al título de Magíster
en Ingeniería Informática**

Asesora:

MSc. Beatriz Susana Acosta Correa

Medellín

Universidad EAFIT

Escuela de Ingeniería

2013

CONTENIDO

| | |
|---|----|
| GLOSARIO..... | 10 |
| RESUMEN | 12 |
| INTRODUCCION | 13 |
| 1. CAPITULO 1..... | 15 |
| 1.1 Sistemas de Información Geográfica - SIG | 15 |
| 1.1.1 Definiciones de SIG..... | 15 |
| 1.1.2 Componentes de un SIG..... | 16 |
| 1.1.3 Funcionalidades de un SIG..... | 18 |
| 1.1.4 Representación de datos geográficos..... | 19 |
| 1.1.5 Comparación entre datos ráster y vectorial | 21 |
| 1.2 Conceptos básicos de geodesia..... | 23 |
| 1.2.1 Mapa..... | 23 |
| 1.2.2 Modelado de la Tierra..... | 24 |
| 1.2.3 Elipsoides de referencia | 25 |
| 1.2.4 Datum | 25 |
| 1.2.5 Coordenadas | 25 |
| 1.2.6 Proyecciones | 26 |
| 1.3 Conceptualización sobre SIG Web..... | 27 |
| 1.3.1 Términos relacionados a SIG Web..... | 29 |
| 1.3.2 Evolución de los SIG Web | 30 |
| 1.3.3 Características de los SIG Web..... | 32 |
| 1.3.4 Funcionalidades de un SIG Web | 33 |
| 1.3.5 Arquitectura SIG Web y componentes..... | 34 |
| 1.3.6 SIG Web actual..... | 37 |
| 1.3.7 Experiencia de usuario en aplicaciones SIG Web | 40 |
| 1.3.8 Mercado y modelos de negocio de los SIG Web..... | 40 |
| 1.4 Servicios Web Geoespaciales..... | 41 |
| 1.5 Tecnologías para la construcción de Aplicaciones SIG Web | 44 |
| 1.5.1 Software libre y propietario | 44 |

| | | |
|-------|---|-----|
| 1.5.2 | Tipos de licencias para productos FOSS..... | 44 |
| 1.5.3 | Software libre para SIG Web: FOSS4GIS y OSGeo..... | 45 |
| 1.5.4 | Herramientas software para construcción de aplicaciones SIG Web | 46 |
| 1.5.5 | Patrones de codificación..... | 53 |
| 1.5.6 | Patrones de Diseño | 58 |
| 1.6 | Metodologías de desarrollo de software..... | 66 |
| 1.6.1 | Metodología tradicional. Rational Unified Process (RUP)..... | 66 |
| 1.6.2 | Metodologías ágiles..... | 69 |
| 2. | CAPÍTULO 2..... | 71 |
| | CONSTRUCCIÓN DE APLICACIONES PILOTO..... | 71 |
| 2.1 | Características de la aplicación piloto | 71 |
| 2.2 | Selección de herramientas software | 72 |
| 2.3 | Desarrollo de aplicación piloto con software libre..... | 78 |
| 2.3.1 | Base de datos..... | 79 |
| 2.3.2 | Publicación de los datos en Geoserver | 81 |
| 2.3.3 | Diseño e implementación de la aplicación..... | 83 |
| 2.4 | Desarrollo de aplicación piloto con software propietario | 94 |
| 2.4.1 | Base de datos..... | 95 |
| 2.4.2 | Publicación de los datos en ArcGISServer | 95 |
| 2.4.3 | Diseño e implementación de la aplicación..... | 96 |
| 3 | CAPITULO 3..... | 101 |
| | LINEAMIENTOS PARA EL DESARROLLO DE APLICACIONES SIG WEB..... | 101 |
| 3.1 | Antes del desarrollo del proyecto | 101 |
| 3.2 | Durante el desarrollo del proyecto..... | 113 |
| 3.3 | Después del proyecto..... | 116 |
| | CONCLUSIONES Y TRABAJOS FUTUROS | 118 |
| | ANEXO A. CODIGO FUENTE DE LAS APLICACIONES PILOTO | 119 |
| | ANEXO B. CONCEPTOS SOBRE LENGUAJE JavaScript..... | 133 |
| | BIBLIOGRAFIA | 146 |

INDICE DE FIGURAS

| | |
|---|----|
| Figura 1. Los seis componentes de un SIG..... | 17 |
| Figura 2. Rasterización de un área. | 20 |
| Figura 3. Tipo vectorial - Polígonos definidos por anillos. | 21 |
| Figura 4. TIN. | 21 |
| Figura 5. Ejemplo de mapa base y temático. | 24 |
| Figura 6. Modelado de la Tierra. | 24 |
| Figura 7. Coordenadas geográficas. | 26 |
| Figura 8. Proyección plana..... | 26 |
| Figura 9. Proyección cónica. | 27 |
| Figura 10. Proyección cilíndrica. | 27 |
| Figura 11. Arquitectura SIG Web. | 28 |
| Figura 12. SIG e Internet y SIG Web..... | 29 |
| Figura 13. Evolución del Web Mapping..... | 32 |
| Figura 14. Arquitectura básica SIG Web. | 34 |
| Figura 15. Distribución de la carga de trabajo. | 36 |
| Figura 16. Mashup del lado del servidor y del lado del navegador..... | 38 |
| Figura 17. Funcionamiento de APIs del lado del navegador y Servicios Web..... | 39 |
| Figura 18. Mercados y modelos de negocio de los SIG Web..... | 41 |
| Figura 19. Roles en una arquitectura de Servicios Web..... | 42 |
| Figura 20. Arquitectura de servicios Web del OGC..... | 43 |
| Figura 21. Free Software and Open Source Software..... | 45 |
| Figura 22. Estructura de proceso RUP..... | 68 |
| Figura 23. Flujo de trabajo y artefactos RUP..... | 68 |
| Figura 24. Clientes para aplicaciones de Web Mapping. | 72 |
| Figura 25. Ranking de los 20 lenguajes de programación utilizados más frecuentemente, de acuerdo a Tiobe para junio de 2011 y octubre de 2012..... | 74 |
| Figura 26. Arquitectura de aplicación con software libre | 78 |
| Figura 27. Base de datos Oracle con datos geográficos del campus EAFIT..... | 81 |
| Figura 28. Geoserver con capas publicadas del campus EAFIT | 83 |

| | |
|---|-----|
| Figura 29. Terminales en Ubuntu con base de datos Oracle, Geoserver y Tomcat | 84 |
| Figura 30. Estructura de una aplicación con ExtJS. | 84 |
| Figura 31. Ejemplo de Main.js. | 85 |
| Figura 32. Ejemplo de ViewLayout.js. | 86 |
| Figura 33. Ejemplo Clase en ExtJS. | 87 |
| Figura 34. Organización de la aplicación con software libre..... | 88 |
| Figura 35. Esquema de la aplicación con software libre | 89 |
| Figura 36. Layout generado con ExtJS y GeoExt..... | 90 |
| Figura 37. Página con mapa de Bloques, con fondo de Google® de calles y satélite..... | 91 |
| Figura 38. Funcionalidad de buscar oficina/aula y localizar bloque..... | 92 |
| Figura 39. Funcionalidad de localizar puntos de interés e identificar | 93 |
| Figura 40. Arquitectura aplicación con software propietario..... | 94 |
| Figura 41. Interfaz de administración de ArcGIS Server 9.3 con datos publicados del Campus EAFIT. | 96 |
| Figura 42. Organización de la aplicación con software propietario | 98 |
| Figura 43. Layout de la página y mapa con imágenes de fondo desde servidor ESRI® | 99 |
| Figura 44. Funcionalidad de búsqueda por bloques..... | 99 |
| Figura 45. Funcionalidad de identificar..... | 100 |
| Figura 46. Código spaghetti. | 102 |
| Figura 47. Modularización básica..... | 110 |
| Figura 48. Encapsulación en OL | 110 |
| Figura 49. Encapsulación en <i>ExtJS</i> | 111 |
| Figura 50. Encapsulación en <i>Dojo</i> | 111 |
| Figura 51. Patrón <i>View Model</i> | 112 |
| Figura 52. Patrón <i>Application Controller</i> | 113 |
| Figura 53. KISS, usabilidad y rendimiento..... | 114 |
| Figura 54. Layout básico | 115 |

INDICE DE TABLAS

| | |
|--|----|
| Tabla 1. Definiciones de SIG según el tipo de usuarios. | 16 |
| Tabla 2. Funcionalidades de un SIG. | 19 |
| Tabla 3. Ventajas relacionadas a la representación ráster y vectorial. | 21 |
| Tabla 4. Comparación ráster vs. vectorial. | 22 |
| Tabla 5. Ventajas y desventajas de ráster y vectorial..... | 22 |
| Tabla 6. Aplicaciones de Web mapping estático. | 30 |
| Tabla 7. Servicios Web vs. APIs del lado del navegador. | 38 |
| Tabla 8. Mercado y modelos de negocio de los SIG Web..... | 40 |
| Tabla 9. Comparación de los conceptos Free software, Open source software y software propietario..... | 44 |
| Tabla 10. Licencias FOSS..... | 44 |
| Tabla 11. Características de OpenLayers. | 46 |
| Tabla 12. Características de GeoExt..... | 46 |
| Tabla 13. Características de Geoserver..... | 47 |
| Tabla 14. Características de Mapserver..... | 48 |
| Tabla 15. Características de Geomajas. | 48 |
| Tabla 16. Características de Mapbender..... | 49 |
| Tabla 17. Características de MapFish..... | 50 |
| Tabla 18. Características de ArcGISServer..... | 51 |
| Tabla 19. Características de PostgrsSQL- Postgis. | 52 |
| Tabla 20. Características de Oracle Locator. | 52 |
| Tabla 21. Ejemplo de separación de comportamiento. | 54 |
| Tabla 22. Comparación RU, XP y Scrum. | 70 |
| Tabla 23. Backlog de los prototipos | 71 |
| Tabla 24. Proyectos de clientes ligeros y Servidores que hacen parte de OSGEo73 | |
| Tabla 25. Proyectos y casos de estudio presentados los eventos Jornadas SIG libre y FOSS4GIS entre 2009 y 2010. | 74 |
| Tabla 26. Cursos ofrecidos de herramientas de SIG libre, en español entre 2010 y 2011. | 75 |
| Tabla 27. Criterios evaluados para selección de herramientas..... | 76 |

| | |
|---|-----|
| Tabla 28. Geoserver vs Mapserver. | 77 |
| Tabla 29. Stack de herramientas para construir aplicación con software libre | 78 |
| Tabla 30. Stack de herramientas para construir aplicación con software propietario | 94 |
| Tabla 31. Niveles de modularización de código | 101 |
| Tabla 32. Ejemplo de lineamiento, para GeoExt y API JavaScript de ArcGIS..... | 105 |
| Tabla 33. Parámetros básicos DASW | 108 |

GLOSARIO

Backend: se refiere a los componentes de la parte servidora que son responsables de la lógica del negocio.

Backlog: lista que contiene los requisitos funcionales y no funcionales del sistema, con la especificación del nivel de prioridad e implicaciones que tiene cada uno de ellos (en caso que así sea).

BluePrints: Java BluePrints es una guía de las mejores prácticas de Sun Microsystems para desarrollo en la Plataforma Java para empresas.

Bounding box: recuadro que especifica cuatro coordenadas minx, miny,maxx y maxy.

CGI: *Common Gateway Interface*.

CVS: *Concurrent Versions System*. Sistema de Control de Versiones, mantiene el registro de todo el trabajo y los cambios en los archivos (código fuente) que forman un proyecto y permite que distintos desarrolladores colaboren.

FOSS: Free and Open Source Software.

FOSS4G: *Free and Open Source Software for Geomatics*. También se refiere a la mayor conferencia sobre software libre para la Geomática organizada por la fundación OSGeo.

Frontend: se refiere a la capa Web que genera las respuestas HTML al navegador.

Geodatabase: es una colección de datasets de diversos tipos que se utiliza en ArcGIS y se administra en una carpeta de archivos o una base de datos relacional. Es la fuente de datos nativa para ArcGIS.

GUI: *Graphical User Interface*. Interfaz gráfica de usuario.

JEE: Java Platform, Enterprise Edition.

JSON: JavaScript Object Notation

Layout: forma de distribuir el contenido en una página Web.

Mapfile: archivo de configuración que tiene como extensión .map. Define los datos a ser usados por la aplicación, tales como: capas, tipos y su configuración; fuente de datos de origen y forma de servirlos datos; leyenda y proyecciones.

Mashup: sistema que integra información de diversas fuentes para brindar nueva información o información enriquecida.

MVC: *Model View Controller*. Patrón modelo vista controlador. La vista despliega los datos, es con la que el usuario interactúa, el modelo maneja el estado de la

aplicación y se comunica con el servidor, el controlador transporta los mensajes desde la vista al modelo.

QGIS - Quantum GIS: SIG de escritorio, *open source* y multiplataforma.

REST: REpresentational State Transfer.

Shapefile: formato de archivo de datos espaciales desarrollado por la compañía ESRI (propietario). Está integrado por, .shp (almacena las entidades geométricas de los objetos), .shx (almacena el índice de las entidades geométricas), .dbf (base de datos, en formato dBASE, donde se almacena la información de los atributos de los objetos), .prj (guarda la información referida al sistema de coordenadas).

SLD: *Styled Layer Descriptor* (SLD), es un esquema XML propuesto por Open Geospatial Consortium como lenguaje estándar para describir el conjunto de capas que dan apariencia a un mapa. Permite definir el estilo visual de cada capa de objetos geográficos que componen el mapa, permitiendo, por ejemplo, representar el color de relleno, tipo y ancho de borde, etc.

SOAP: Simple Object Access Protocol.

UML (*Unified Modeling Language*): lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

WFS: *Web Feature Service*.

WMS: *Web Map Service*.

RESUMEN

El proyecto de generar una serie de lineamientos que orienten el desarrollo de aplicaciones SIG Web, surgió en el CentrolG (Laboratorio de SIG), con el apoyo de la dirección de investigación de EAFIT. La propuesta surgió a raíz de dos casos: la construcción del Campus Universitario Georreferenciado (2009) y del prototipo del Sistema de Movilidad de Medellín (2010), en los cuales se enfrentaron dificultades a lo largo de las etapas de diseño, construcción y producción. A partir de esto, se vio la oportunidad de recoger el conocimiento y las lecciones aprendidas, para divulgarlas a los profesionales del campo de las tecnologías de la información y las comunicaciones, especialmente para aquellos del medio local y nacional. Con el objetivo de que los lineamientos tuvieran un buen fundamento se planteó una metodología que abarcara en primer lugar el estudio teórico de buenas prácticas en ingeniería de software y conceptos de SIG. En segunda instancia, se realizó un trabajo práctico, mediante la construcción de pilotos, focalizando la atención en el cliente (*frontend*), uno con software propietario y otro con software libre. Entre las herramientas seleccionadas están Geoserver, GeoExt, ArcGIS Server y el API JavaScript de ESRI. Los lineamientos se plantearon en tres momentos, antes, durante y después del desarrollo. Entre los propuestos se encuentran: la conformación adecuada del equipo humano, parámetros para la selección de tecnologías, la arquitectura como punto de partida y la modularización de una aplicación como eje para su evolución.

Palabras clave: SIG WEB, DESARROLLO SOFTWARE, BUENAS PRÁCTICAS, SOFTWARE LIBRE, SOFTWARE PROPIETARIO.

INTRODUCCION

Existe hoy en día una creciente necesidad de aplicaciones de Sistemas de Información Geográfica (SIG) en Web, ya que son muchas áreas del saber que requieren el uso de datos geoespaciales para cumplir con mayor acierto sus procesos, tal como la gestión pública, medioambiental, ingenieril, entre otras. También son solicitadas por comunidades de usuarios que están dispersos y por organizaciones o entidades que precisen compartir e integrar datos georreferenciados para realizar diferentes tipos de análisis espacio-territoriales y ayudar en la toma de decisiones. De igual forma, los ciudadanos comunes se interesan cada vez más en herramientas que les permitan visualizar mapas y obtener información de su interés (rutas de transporte, estado del tráfico, sitios turísticos, localización de direcciones). Sin embargo, muchos de los productos software desarrollados no satisfacen plenamente los objetivos para los que fueron diseñados, presentando múltiples defectos cuando son puestos en uso. Fallas y errores que son originados en diferentes puntos del proceso de desarrollo: desde requisitos incompletos o mal entendidos, arquitecturas que pasan por alto calidades sistémicas vitales (*performance*, disponibilidad, mantenibilidad, usabilidad, etc), tecnologías inadecuadas o empleadas de manera equivocada, hasta despliegues en hardware insuficiente para soportar las aplicaciones.

A partir del recorrido y el quehacer diario en el Centro IG - Laboratorio de Sistemas de Información Geográfica de la universidad EAFIT, se evidencia que a nivel local, regional y nacional, hay poco conocimiento y falta de experiencia en la creación de aplicaciones SIG en ambiente Web, preguntas del tipo ¿cómo publico un mapa en Internet?, ¿cómo incluyo un mapa en una página Web?, ¿qué herramientas puedo utilizar?, ¿dónde se guarda un mapa? son comunes de escuchar, por lo cual se hace necesario divulgar y contribuir de manera práctica en la formación académica en este campo.

Vale citar dos experiencias que motivaron en gran medida la realización de esta investigación: la primera, se remite al año 2006, en el que el Laboratorio de SIG y el Centro de Informática iniciaron el proyecto Campus Georreferenciado, el cual tenía por objetivo la creación y gestión de información espacial de EAFIT. En la primera fase se levantó la cartografía de la universidad, cumpliendo con rigor la métrica y haciendo énfasis en la calidad de los datos. Al estudiar las múltiples utilidades que se le podían dar a los datos generados, se optó por desarrollar el mapa interactivo, una aplicación Web, que tenía por misión, orientar a las personas sobre la ubicación de bloques, empleados, sitios de interés, dependencias y trazar rutas entre dos puntos. En el año 2009 se hizo su lanzamiento, la puesta en producción dejó al descubierto varios problemas, entre ellos la demora en tiempo de respuesta e interfaces de usuario poco intuitivas. La segunda, se enmarca entre los años 2009 - 2010, los estudiantes de especialización en Desarrollo de Software de EAFIT elaboraron un proyecto que consistía en la construcción de una aplicación SIG Web para la Secretaría de Movilidad (Tránsito y Transporte) de la ciudad de Medellín. La premisa inicial era utilizar herramientas de código abierto (*open source*). Dentro de las

funcionalidades que debía exponer el sistema estaban: consultar paraderos autorizados y rutas de transporte (cerca a un punto señalado en el mapa, por barrio y búsqueda alfanumérica), geocodificación, consultar sitios de interés, gestión de incidentes y ubicación de brigadas/agentes de tránsito. Fue un buen trabajo de acercamiento y experimentación con tecnologías, al que se le dedicó estudio y tiempo, pero infortunadamente no pudo ser sacado a producción por problemas en la arquitectura propuesta y en la gestión del proyecto.

A partir de las situaciones anteriores surgió la idea de recoger las experiencias y conocimientos adquiridos, con el fin de darlos a conocer a la comunidad académica del área. Sin embargo, se vio una oportunidad mayor al ampliar la visión, al proponer una serie de lineamientos o guías para orientar la construcción de aplicaciones SIG Web, dirigidos especialmente a profesionales en el campo de las tecnologías de la información y las comunicaciones y de esta forma contribuir en la formación respecto al manejo de datos geoespaciales y abrir posibilidades en éste ámbito en plena expansión.

Los lineamientos formulados están soportados tanto por las bases teóricas y buenas prácticas en el campo de la ingeniería de software y los SIG, como por los aprendizajes derivados de la construcción de aplicaciones piloto, las cuales permitieron obtener perspectivas a nivel general (de todo el proceso) y detallado (en las etapas de análisis, diseño e implementación). Se tomaron dos herramientas representativas para su elaboración, una de software libre y otra de tipo licenciado.

A continuación se describe la estructura de los capítulos de este trabajo:

Capítulo 1, contiene las bases teóricas en cuanto a Sistemas de Información Geográfico, datos geográficos, geodesia, SIG Web y tecnologías para su construcción.

Capítulo 2, comprende el proceso de elaboración de las aplicaciones piloto, la selección de las herramientas, instalación, configuración, adecuación de los datos, esquema de la arquitectura y codificación.

Capítulo 3, recoge los lineamientos para la construcción de aplicaciones SIG Web. Conclusiones y trabajos futuros.

Anexo A contiene parte del código fuente de las aplicaciones piloto.

Anexo B explica conceptos importantes del lenguaje JavaScript.

1. CAPITULO 1

MARCO CONCEPTUAL SOBRE APLICACIONES SIG WEB

Este capítulo recoge la conceptualización relevante para la comprensión de las temáticas alrededor del desarrollo de aplicaciones SIG Web, es importante que todos los agentes involucrados tengan un nivel de conocimiento general suficiente, aunque cada uno tenga una especialización diferente, de esta manera se facilitarán las sinergias entre el personal del mundo SIG y el que está relacionado con las tecnologías de la información y las comunicaciones. Los apartados 1.1 Sistemas de Información geográfica - SIG y 1.2 Conceptos básicos de Geodesia ofrecen un panorama de la naturaleza de los SIG y los datos geoespaciales. Los numerales 1.3, SIG Web y 1.4 Servicios Web Geoespaciales explican las particularidades de este tipo de aplicaciones. Las últimas secciones, 1.5 y 1.6, se concentran en el proceso de desarrollo de software, al cubrir tecnologías y metodologías.

1.1 SISTEMAS DE INFORMACIÓN GEOGRÁFICA - SIG

1.1.1 Definiciones de SIG.

La concepción de lo que es un SIG ha evolucionado desde los años 60's, algunas definiciones que dan cuenta de esto son:

Un SIG es un sistema de hardware, software y procedimientos elaborados para facilitar la obtención, gestión, manipulación, análisis, modelado, representación y salida de datos espacialmente referenciados, para resolver problemas complejos de planificación y gestión (NCGIA, 1990).

Según Korte (2001), un SIG es un sistema que integra tecnología informática, personas e información geográfica, y cuya principal función es capturar, analizar, almacenar, editar y representar datos georreferenciados

Para Longley et al. (2010), los SIG son sistemas basados en computador, para el almacenamiento y procesamiento de información geográfica. Son herramientas que mejoran la eficiencia y efectividad en el manejo de información sobre objetos geográficos y eventos. Pueden ser utilizados para llevar a cabo múltiples tareas, incluyendo almacenar grandes cantidades de información geográfica en bases de datos, realizar operaciones analíticas en una fracción del tiempo del que tomaría hacerlas manualmente y automatizar el proceso de creación de mapas.

La etiqueta SIG se ha dado a muchas cosas: desde la colección de herramientas software, las representaciones digitales de varios aspectos del mundo geográfico en forma de conjuntos de datos, la comunidad de personas que los emplean para diversos propósitos, hasta la actividad de utilizarlos para resolver problemas y en aspectos de ciencia avanzada.

La siguiente tabla, presenta definiciones de SIG, de acuerdo con el público usuario

Tabla 1. Definiciones de SIG según el tipo de usuarios. Tomado de (Longley et al., 2010)

| Definición | Grupo de usuarios |
|---|---|
| Contenedor de mapas en forma digital | Público general |
| Herramienta computacional para solucionar problemas geográficos. | Encargados de la toma de decisiones, y planeación, comunidad/grupos |
| Sistema espacial para la toma de decisiones | Gestores científicos, investigadores de operaciones. |
| Inventario mecanizado de objetos geográficos distribuidos e instalaciones. | Gestores de servicios públicos, movilidad/transporte |
| Herramienta que revela lo que de otra forma es invisible | Científicos, investigadores |
| Herramienta para ejecutar operaciones sobre datos geográficos, que de hacerlas a mano resultarían muy tediosas, costosas o inexactas. | Gestores de recursos, planeadores |

1.1.2 Componentes de un SIG.

Longley et al.,(2010), establecen seis componentes, divididos en dos grupos:

- Red: es parte fundamental, ya que permite comunicación rápida y compartir información con gran cantidad de usuarios. En la actualidad, los SIG se basan fuertemente en Internet e intranets de tipo corporativo, gubernamental y militar, debido a que es una plataforma establecida, ampliamente utilizada y con estándares aceptados para interactuar con información de muchos tipos, además de enlazar usuarios distribuidos de forma económica. De otro lado, las redes inalámbricas y los dispositivos móviles han posibilitado el acceso en tiempo real a los servicios geográficos (i.e. rutas, mapas). Un factor de alta importancia fue el nacimiento y adopción de tecnologías Web 2.0, tales como AJAX (Asynchronous Javascript and XML) y el desarrollo de APIs, las cuales mejoraron significativamente la usabilidad y facilidad de creación de las aplicaciones SIG Web.

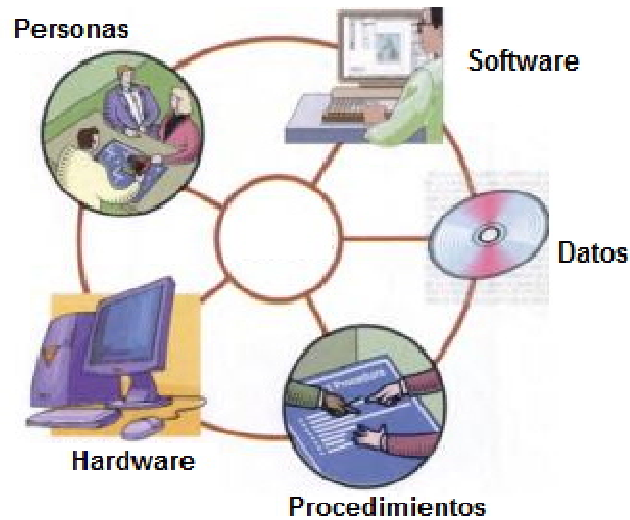


Figura 1. Los seis componentes de un SIG. Adaptado de (Longley et al., 2010)

- Otros, constituidos por:

Hardware: son los dispositivos con los que el usuario interactúa para llevar a cabo cualquier operación SIG. Tradicionalmente era un computador de escritorio, hoy en día se han extendido a portátiles, PDAs, teléfonos móviles, entre otros.

Software: que se ejecuta localmente en la máquina del usuario. Puede ir desde un navegador Web estándar hasta paquetes completos con una amplia gama de funcionalidades.

Datos: son representaciones digitales de aspectos seleccionados de un área de la superficie (o cercana) de la Tierra, que se emplean para resolver un problema o con un propósito científico. Una base de datos puede ser construida para un proyecto o puede estar en constante mantenimiento, alimentada por transacciones diarias. El tamaño puede variar, desde unos pocos megabytes a varios terabytes.

Procedimientos: un SIG requiere de gestión, se deben establecer procesos, informes, puntos de control y otros mecanismos para asegurar que las actividades se dirijan a cumplir con las necesidades, dentro del presupuesto estimado, se mantenga alta calidad y en general que se apunte a los objetivos de la organización.

Personas: diseñan, programan, mantienen, suministran datos e interpretan resultados. El personal SIG tiene diferentes habilidades, dependiendo del rol que desempeñe. Casi todos cuentan con conocimientos básicos para trabajar con datos geográficos (fuentes de datos, escalas, proyecciones, precisión, exactitud, productos software) y establecen redes con conocidos de la comunidad SIG.

Olaya (2011), cita una forma diferente de organizar los componentes:

- Datos, constituyen el corazón de un SIG, y los demás componentes se basan en ellos para ejercer su papel dentro del sistema. Se deben considerar varios aspectos: gestión y almacenamiento en bases de datos, ya que por naturaleza son voluminosos y su tendencia es crecer; origen de los datos, distintas formas en que la información geográfica puede recogerse y su posterior integración; la calidad que deben cumplir.
- Procesos, el análisis es una funcionalidad básica de los SIG y unido a la capacidad de procesamiento computacional permite obtener resultados a partir de los datos espaciales, son diversos los campos en los cuales puede ser aplicado. Las formulaciones que pueden aplicarse van desde procesos sencillos (consultas simples o mediciones) a otros de mayor complejidad (manejo de variables numerosas), siendo la estadística un apoyo de los SIG para realizar esta tarea.
- Visualización, es la forma principal de trabajar con la información geográfica, es intuitiva, fácil de comprender y a la que las personas están más acostumbradas. El SIG almacena datos alfanuméricos, por esto debe ser capaz de representarlos o traducirlos de manera gráfica. El SIG debe incorporar capacidades de creación y diseño cartográfico así como su impresión. Además de las representaciones tradicionales, los SIG ofrecen otras más avanzadas como las tridimensionales, de ambas formas la interactividad que ofrece a los usuarios es elevada.
- Tecnología, respecto a la plataforma hardware en la cual se llevan a cabo tareas SIG, se distinguen los PCs y servidores. En cuanto a periféricos se mencionan las tabletas digitalizadoras, plotters, scanners, GPS. El software opera y manipula datos, las aplicaciones clásicas permiten visualizar, gestionar y analizar la geoinformación. Existen herramientas de mayor especialización que se centran en alguna de las tareas o son componentes que pueden reutilizarse en otras aplicaciones.
- Factor organizativo: los SIG requieren de una correcta organización entre los elementos que lo constituyen. De especial importancia la relación entre las personas encargadas del SIG y la interacción de todos los componentes con los datos, ya que la información es un activo compartido que debe gestionarse. A partir de esto, surgen diversos perfiles alrededor del funcionamiento del SIG, tales como el usuario final o clásico que pueden ser no expertos, los administradores de bases de datos, arquitectos empresariales y de software.

1.1.3 Funcionalidades de un SIG.

La siguiente tabla resume, a manera de pregunta y ejemplo las funcionalidades de un SIG:

Tabla 2. Funcionalidades de un SIG. Elaborado con información de GISLAN

| Funcionalidad | Ejemplo |
|--|--|
| Localización: Qué hay en...? | ¿Quién es el propietario de la parcela X? |
| Condición: ¿Dónde sucede tal cosa...? | ¿Donde se encuentran las fuentes de recursos más cercanas? |
| Tendencias: ¿Qué ha cambiado? | ¿Cómo ha evolucionado nuestro mercado potencial en los últimos años? |
| Patrones: ¿Qué patrón espacial existe en...? | ¿Dónde y a qué hora se producen los atascos de tráfico? |
| Modelos: ¿Qué ocurriría si...? | ¿Cuál sería el impacto ecológico de la implantación de un parque eólico en este lugar? |
| Rutas: ¿Cuál es el camino óptimo? | Calcular el camino óptimo (corto, rápido, barato...) entre vario puntos. |

1.1.4 Representación de datos geográficos.

Se distinguen tres tipos de datos: ráster, vector y TIN.

- Ráster: Es una matriz bidimensional compuesta por píxeles, en donde cada píxel posee un valor numérico que se representa visualmente con un color. Entre los tipos ráster se encuentran: imágenes satelitales, fotos aéreas, cartografía digitalizada. Por ejemplo: en una foto aérea, un píxel verde se asocia con vegetación, en una imagen infrarroja satelital, el valor puede relacionarse con el nivel de humedad.

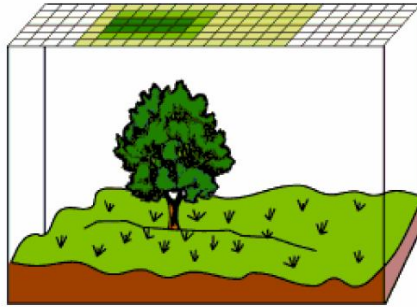


Figura 2. Rasterización de un área. Tomada de (González et al., 2011)

Tipos de datos para los que los ráster son adecuados:

- Datos discretos (i.e. uso del suelo)
- Datos continuos (i.e. elevación, información espectral)
- Imágenes (i.e. fotos, mapas escaneados)

Usos de los ráster:

- Mapa base: primera capa sobre la que se superponen capas vectoriales. Por ejemplo: foto satelital (ráster) + calles (vectorial).
 - Mapa de superficie: representa parámetro que varía continuamente sobre el terreno. Por ejemplo: elevación, humedad, lluvias, densidad de población.
 - Digital Elevation Model - DEM: es un tipo de ráster muy utilizado en donde el valor indica la altura del punto. A partir de un DEM pueden derivarse curvas de nivel.
 - Mapa temático: clasificación de los píxeles en clases o categorías. Por ejemplo: en un ráster de uso de la tierra, las clases pueden ser actividades económicas (agricultura, ganadería, forestación, etc).
 - Ráster como atributo: se utilizan fotografías de un objeto geográfico como un atributo, que puede mostrarse junto con los alfanuméricos.
- Vectorial: Representan de forma geométrica los objetos geográficos en forma precisa y compacta, utilizando puntos, líneas y polígonos. Se puede emplear este tipo para entidades como edificios, parcelas, tuberías, calles, entre otros.

Tipos de datos vectoriales:

- Punto: coordenadas (x,y).
- Multipunto: conjunto de puntos.
- Línea o polilínea: secuencia de segmentos conectados que no se interceptan (caminos). Cada segmento se define por dos coordenadas (x,y). Los puntos consecutivos pueden definir un segmento de recta o una curva.
- Polígono: conjunto de anillos. Cada anillo es un serie cerrada de coordenadas (x,y) que define un área. Permite representar áreas con "huecos" o "islas"



Figura 3. Tipo vectorial - Polígonos definidos por anillos. Tomada de (González et al., 2011)

- Triangular ó TIN: una Red Irregular Triangular (Triangular Irregular Network - TIN) es un modelo eficiente y preciso para representar superficies continuas, construido en base a nodos distribuidos en forma irregular (con coordenadas x,y,z) y líneas conectoras que forman un red de triángulos. Se utiliza en: análisis de elevación, pendientes, volúmenes, visibilidad desde un punto, entre otras aplicaciones. También permiten visualización 3D Se suelen construir a partir de un DEM (Modelos digitales de elevación), eligiendo puntos de interés en forma algorítmica.

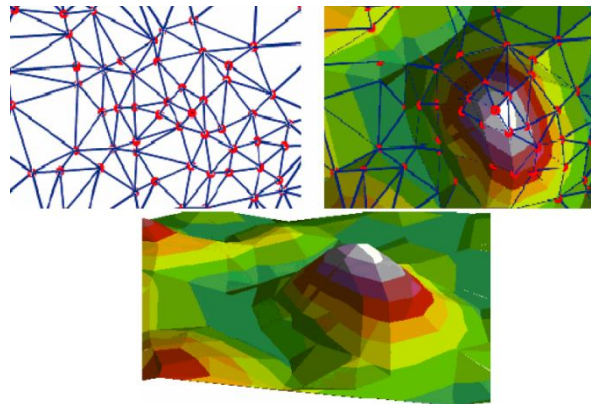


Figura 4. TIN. Tomada de (González et al., 2011)

1.1.5 Comparación entre datos ráster y vectorial

El paralelo entre los dos tipos de datos se encuentra en las siguientes tablas, de acuerdo con autores y criterios distintos:

Tabla 3. Ventajas relacionadas a la representación ráster y vectorial. Tomada de (Longley et al. 2010)

| Característica | Ráster | Vectorial |
|----------------------|--------------------------------|--|
| Volumen de los datos | Depende del tamaño de la celda | Depende de la densidad de los vértices |
| Fuentes de datos | Percepción remota, imágenes | Social y datos ambientales |
| Aplicaciones | Recursos, ambiental | Social, económico, administrativo |

| | | |
|------------|---|---|
| Software | Procesamiento de imágenes, SIG específico para ráster | Cartografía automatizada, SIG específico para datos vectoriales |
| Resolución | Fijo | Variable |

Tabla 4. Comparación ráster vs. vectorial. Tomado de (Olaya, 2011)

| Característica | Ráster | Vectorial |
|---------------------------|---|--|
| Planteamiento | Los análisis responden a preguntas sobre qué y cómo. | Se enfoca en contestar a interrogantes sobre localización: dónde |
| Precisión | Está limitada por el tamaño de celda. Existen también imprecisiones en las formas (se restringen a ángulos rectos, por ser cuadrados) | El detalle de las formas es más fiel al del objeto real representado. |
| Volumen de almacenamiento | Requiere de grandes volúmenes. | Es de menor exigencia. |
| Complejidad | La implementación de algoritmos de análisis es sencilla, pero son costosos en términos de tiempo. | La implementación de algoritmos es de mayor dificultad, debido a su irregularidad espacial y son costosos en términos de capacidad de cálculo. |

Tabla 5. Ventajas y desventajas de ráster y vectorial. Tomado de (Buckley, 1997)

| | Ráster | Vectorial |
|----------|--|--|
| Ventajas | <ul style="list-style-type: none"> -La localización geográfica de cada celda está implícita en la matriz. Los únicos que almacenan coordenadas geográficas son los puntos de origen (i.e esquina inferior izquierda). -Las técnicas de análisis de datos son fáciles de programar y rápidas de ejecutar. -Son adecuados para modelado | <ul style="list-style-type: none"> -Datos pueden ser representados en su resolución y forma original sin generalización. -Los gráficos, usualmente son estéticamente más agradables. -La exactitud de la localización geográfica de los datos se mantiene. -Permite una codificación eficiente de la topología, dando como resultado operaciones más |

| | | |
|-------------|--|---|
| | matemático y análisis cuantitativo. | eficientes, como por ejemplo los análisis de proximidad y de redes. |
| Desventajas | <p>-El tamaño de la celda determina la resolución a la que son representados los datos.</p> <p>-El procesamiento de los atributos puede ser pesado, si hay gran cantidad de datos. Los ráster, inherentemente reflejan un atributo o característica para un área.</p> <p>-Si los datos deben convertirse a forma vectorial, se pueden presentar problemas con la integridad de los datos, debido a la generalización y escogencia inapropiada del tamaño de celda.</p> | <p>-La localización de cada vértice necesita almacenarse explícitamente.</p> <p>-Para análisis efectivos, los datos vectoriales deben ser convertidos a una estructura topológica, en general, este proceso es intensivo y requiere de limpieza de datos. Cuando hay edición o actualización se debe reconstruir la topología.</p> <p>-Los algoritmos de manipulación y funciones de análisis son complejos y pueden ser intensivos en procesamiento. Esto limita la cantidad de datos.</p> |

1.2 CONCEPTOS BÁSICOS DE GEODESIA

1.2.1 Mapa.

Según International Cartographic Association, es la representación, a escala y sobre un medio plano, de una selección de entidades sobre la superficie de la Tierra. La noción de "mapa" se utiliza en matemáticas y computación como una transferencia de información de una forma en otra, de la misma forma que un cartógrafo transfiere información desde la superficie de la Tierra a un papel.

La elaboración de un mapa implica:

- Selección de entidades.
- Clasificación de entidades (i.e. calles, vías de tren, autopistas, etc.)
- Escala
- Simplificaciones (i.e. polilíneas en lugar de curvas, eliminación de irregularidades, etc.)
- Exageraciones (i.e. edificios que no se muestran a su escala real).
- Simbolizaciones (i.e. avión para aeropuertos, color azul para el agua, etc.)
- Convenciones cartográficas (i.e. norte arriba)

Tipos de mapas:

- Temáticos: son aquellos que están diseñados para mostrar conceptos particulares sobre un tema
- Base: se utilizan como marco para mostrar otra información.

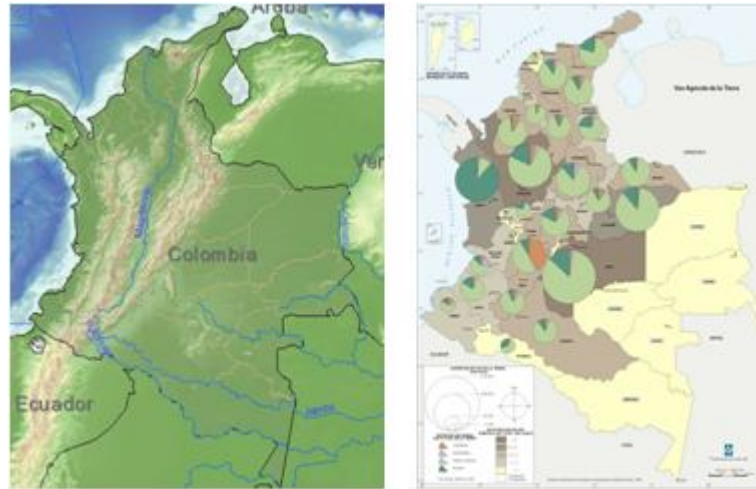


Figura 5. Ejemplo de mapa base (izq.) y temático (der.). Tomada de <http://mapascolombia.igac.gov.co>

1.2.2 Modelado de la Tierra

La Tierra tiene forma elipsoidal, ligeramente achatada en los polos. Son tres modelos que se utilizan para construir mapas: la esfera, el elipsoide y el geoide.

- La esfera y el elipsoide son volúmenes geométricos (el elipsoide se aproxima mejor)
- El geoide es un volumen irregular que se obtiene mediante mediciones. Es la mejor aproximación.

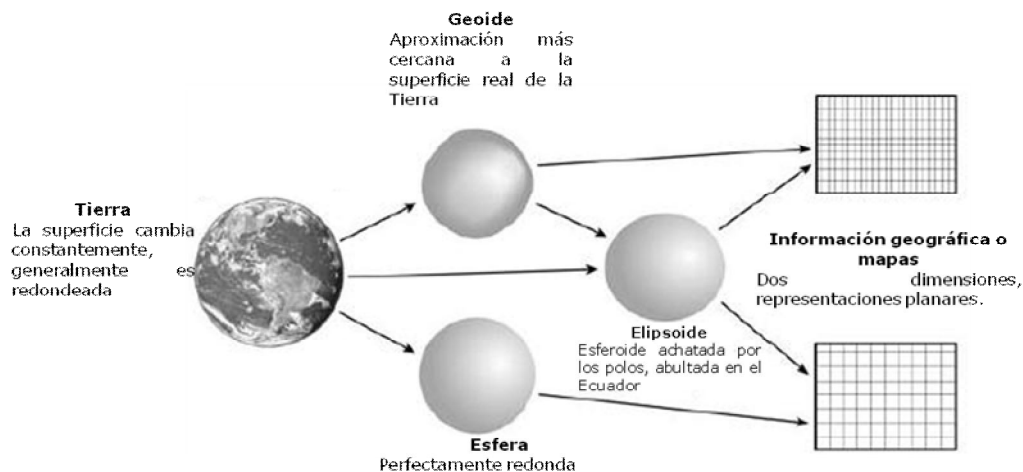


Figura 6. Modelado de la Tierra. Adaptada de (González et al., 2011)

1.2.3 Elipsoides de referencia

Se definen mediante su radio semi-mayor (ecuatorial) y semi-menor (polar), o por la excentricidad (relación entre radio semi-mayor y achatamiento en los polos). Se los denomina generalmente por su autor y año (i.e. Clarke 1880). Uno de los más utilizados es el WGS84

1.2.4 Datum

El datum define el conjunto de parámetros que permite poner en relación un elipsoide con un geoide. Los parámetros incluyen el punto de contacto, los parámetros del elipsoide, el sistema de referencia, etc.

Existen datums simples de tierra plana hasta complejos que describen tamaño, forma, orientación, campo gravitacional y velocidad angular de la Tierra. Diferentes países, zonas y organizaciones utilizan distintos datums.

Las coordenadas dadas en un datum erróneo pueden tener errores de cientos de metros.

Tipos de datums:

- Horizontales, definen la relación entre la Tierra y las coordenadas horizontales (como latitud y longitud). Ejemplos: NAD27, ED50.
- Verticales, definen superficies de nivel, en base a mediciones sobre el nivel del mar y redes de nivelación o mediciones gravitacionales.
- Completos, son tanto horizontales como verticales. Ejemplo: WGS84.

1.2.5 Coordenadas

El proceso de asociar coordenadas absolutas a un punto de la superficie de la Tierra se llama georreferenciación, se utilizan coordenadas planas o geográficas.

Las coordenadas planas cartesianas utilizan dos ejes ortogonales igualmente escalados, y cada punto del plano se define por sus coordenadas (x,y), dependen de la proyección cartográfica utilizada. Otro tipo de coordenadas planas son las coordenadas polares.

Las coordenadas geográficas (o geodésicas) se definen por el par (latitud,longitud).

- Latitud: medida de la distancia angular (entre 0 y 90°) de un punto entre el Ecuador su polo más cercano (i.e. Colombia se sitúa entre las latitudes 12N y 4 S).
- Longitud: medida de la distancia angular (entre 0 y 180°) de un punto y el Meridiano de Greenwich (i.e. Colombia se sitúa entre las longitudes 66 y 79 W).

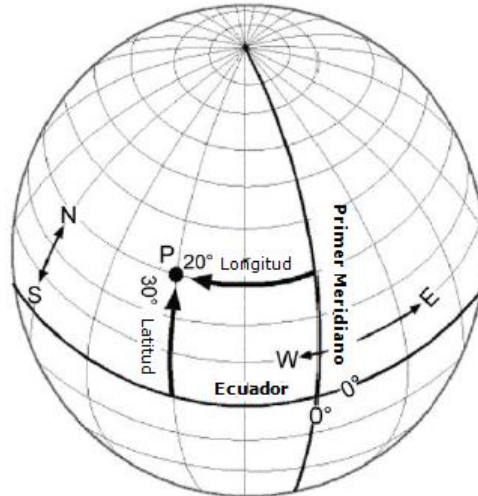


Figura 7. Coordenadas geográficas. Adaptada de (González et al., 2011)

1.2.6 Proyecciones

Es una relación biunívoca entre los puntos de la superficie curva de la Tierra y los puntos de una superficie plana (mapa). Se utiliza una malla de meridianos y paralelos para ubicar los puntos. Esto permite pasar de coordenadas geográficas a coordenadas planas. Las proyecciones distorsionan ángulos, direcciones, áreas, formas y distancias. Cada proyección se comporta mejor o peor respecto a alguna de estas propiedades.

Existen 3 tipos de proyecciones básicas:

- Planas o azimutales, se hace pasar un plano tangente por un punto de contacto con el globo o secante en una circunferencia. Se proyectan los puntos en el plano, transformando las coordenadas geográficas en coordenadas polares, con el origen en el punto de contacto o en el centro.

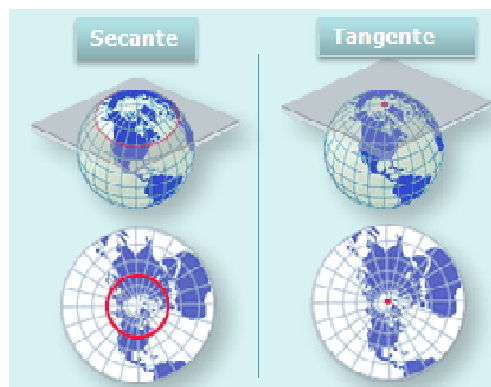


Figura 8. Proyección plana. Tomada de (González et al., 2011)

- Cónicas, se hace pasar un cono tangente al globo en una circunferencia (como un paralelo) o secante en dos circunferencias. Se proyectan los puntos en el

cono, transformando las coordenadas geográficas en coordenadas polares, con el origen en vértice del cono.

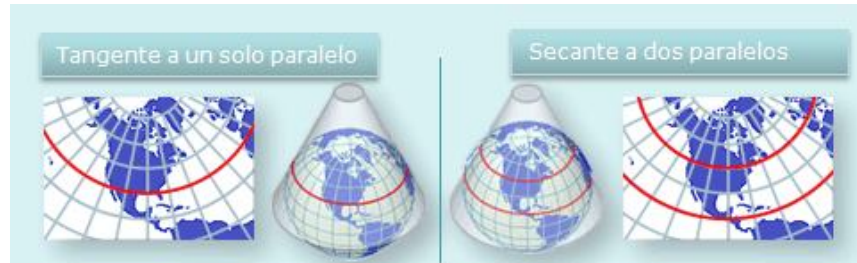


Figura 9. Proyección cónica. Tomada de (González et al., 2011)

- Cilíndricas, se hace pasar un cilindro tangente al globo en una circunferencia o secante en dos circunferencias. La proyección puede ser ecuatorial, transversa u oblicua. En el caso ecuatorial, se utilizan coordenadas cartesianas con el x igual a la longitud y la y en función de la latitud.

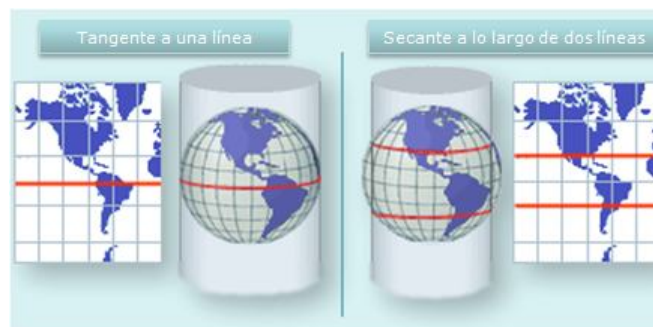


Figura 10. Proyección cilíndrica. Tomada de (González et al., 2011)

También existen proyecciones modificadas que se basan en las anteriores y proyecciones no geométricas. Otra clasificación de proyecciones tiene en cuenta las propiedades que se preservan:

- Conformes, preservan los ángulos (y forma en regiones pequeñas o medianas).
- Equivalentes, preservan el área.
- Equidistantes, preservan la distancia entre puntos dentro de determinadas líneas.
- De compromiso, no preservan ninguna propiedad, pero tampoco presentan grandes distorsiones.

1.3 CONCEPTUALIZACION SOBRE SIG WEB

Según Shen, Cheng y Gong (2008), un SIG Web es una aplicación que es accesible a través de un navegador. Hace posible a los usuarios tener acceso a funcionalidades SIG básicas como localizar, obtener direcciones, hacer zoom e imprimir mapas. En otras palabras, SIG Web significa que la información

geoespacial se puede publicar, hacer búsquedas, analizar, desplegar y procesar sobre Internet.

Para Dragicevic (2004), las aplicaciones SIG Web mejoran las capacidades de los usuarios en tres formas distintas. La primera es el acceso a los datos espaciales y otra información. Con un ambiente basado en Web, el SIG se convierte en interactivo, dinámico y accesible a grandes grupos de usuarios como herramienta de comunicación visual. La segunda es la exploración de datos espaciales y geovisualización que ofrece a las personas de negocios apoyo para tomar decisiones. La tercera es procesamiento, análisis y modelamiento de datos espaciales.

De acuerdo a You, Liu y Lin (2007), un mapa en Web, es un servicio en línea sobre Internet, que provee mapas a los usuarios y los ayudan en la búsqueda y navegación de información espacial, como localizar diferentes lugares y trazar rutas.

Peng y Tsou (2003) definen un SIG Web como un SIG distribuido a través de una red de computadores para integrar, diseminar y comunicar información geográfica a través de la WWW.

Según Fu y Sun (2011), un SIG Web es un tipo de sistema de información distribuido. De manera concisa, se considera que es cualquier SIG que utiliza tecnología Web para comunicar sus componentes.

Chivite y Zwaap (2011), señalan que una aplicación Web es un aplicación a la que se puede acceder a través de una red (intranet o internet). En el caso de los SIG se denomina aplicación de *Web mapping*, que es utilizada dentro de un navegador. Generalmente no tiene todas las características de un SIG tradicional, pero si un propósito específico. Una aplicación de *Web mapping* está conformada por: mapas (contenido) y funcionalidad.

El siguiente gráfico ilustra una arquitectura SIG Web simple, que cuenta con un servidor Web de aplicaciones y un cliente, que puede ser un navegador, una aplicación de escritorio o móvil, la comunicación se realiza vía HTTP.

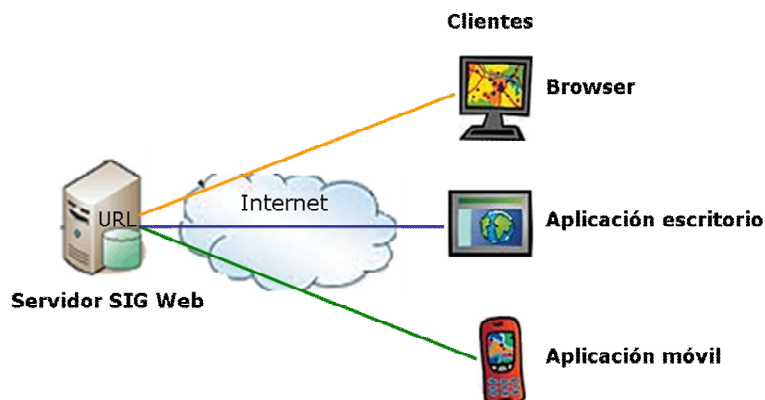


Figura 11. Arquitectura SIG Web. Adaptado de (Fu y Sun, 2011)

1.3.1 Términos relacionados a SIG Web

En las definiciones anteriores y en la literatura se encuentran algunos términos, que aunque parecidos, tiene sutiles diferencias:

- SIG en Internet (*Internet GIS*), es utilizado por los autores Peng y Tsou (2003). Primero aclaran que Internet es una red, compuesta de múltiples redes, dispersas geográficamente, conectada a través de dispositivos y un conjunto de protocolos de comunicaciones. La World Wide Web (WWW) es una aplicación de red soportada sobre http, que corre sobre Internet. Un SIG en Internet se refiere al uso de Internet como medio para intercambiar datos, realizar análisis y presentar resultados, utilizando cualquiera de los servicios ofrecidos por ella. Los SIG Web son entonces un subconjunto de los SIG en Internet, pero son la forma más ampliamente usada. Los mismos autores mencionan el concepto de SIG distribuido, el cual representa un marco más amplio, que incluye los SIG en Internet y los SIG móviles (que emplean las redes inalámbricas). En el siguiente gráfico se observa esta relación:

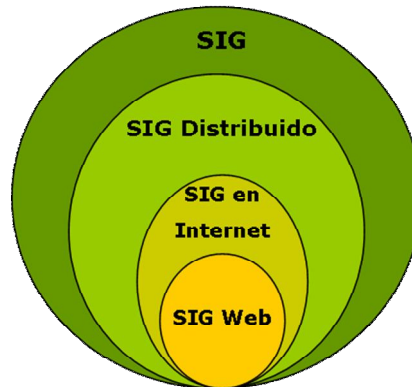


Figura 12. SIG e Internet y SIG Web. Adaptada de (Fu y Sun,2011)

- Web Geoespacial o GeoWeb: es empleado en dos contextos, el primero como combinación de información geoespacial con abstracta (no geoespacial, i.e. fotos, videos, noticias). Se relaciona con *geotagging* (análisis de contenido para identificar referencias geográficas) y *geoparsing* (lectura de documentos y páginas Web en lenguaje humano para identificar términos geográficos y referencias mediante técnicas de procesamiento de lenguaje natural, -NPL). Y el segundo, para referirse a un SIG de tipo global emergente.
- Web Mapping: "The SDI Cookbook v2.0" (2004), lo emplea para hacer referencia a la visualización de datos geoespaciales utilizando la Web. Incluye la presentación de mapas de propósito general para desplegar lugares y accidentes geográficos, así como herramientas cartográficas interactivas y personalizadas. Su intención es representar la información de manera rápida y fácil para la mayoría de los usuarios.

1.3.2 Evolución de los SIG Web

Peng y Tsou (2003), relatan este proceso: la década de los noventa vio el nacimiento del "Internet Mapping", lo que constituyó el primer paso hacia los SIG en Web. Se reconocen tres grandes etapas:

Etapa I: publicación de mapas estáticos: en este tipo de aplicación se insertan imágenes de mapas (formatos gif, jpeg, PNG, PDF) en documentos HTML. También se encuentran los llamados mapas interactivos, en los cuales dependiendo del lugar en el que el usuario da *click* sobre la imagen, se enlaza a diferentes páginas HTML. La arquitectura software es de dos capas: cliente (navegador Web) / servidor, comunicados mediante HTTP.

Etapa II: Web Mapping estático, esta clase de aplicaciones se caracterizan por construir mapas de acuerdo con las peticiones enviadas por el cliente, el servidor elabora una imagen y la retorna al navegador para que las muestre en una página HTML. Adicional al mapa, se presenta información y se pueden realizar algunos análisis. El usuario se ve limitado ya que sólo interactúa a través de formularios y no directamente con el mapa. La arquitectura es de tres capas: cliente / servidor unido a un middleware o CGI. Los CGIs presentan dos grandes inconvenientes: el no mantener el estado entre conexiones *-stateless-*, siendo un obstáculo para algunas operaciones SIG que lo requieren. El otro problema está relacionado con la carga del programa en memoria por cada petición, lo que conlleva a limitaciones en el desempeño. El siguiente cuadro contiene algunos desarrollos utilizando esta clase de tecnología:

Tabla 6. Aplicaciones de Web mapping estático. Elaborada con información de (Peng y Tsou, 2003)

| Nombre | Promotor | Año | Características |
|-----------------|----------|------|--|
| PARC Map Viewer | Xerox | 1993 | Fue el primer visualizador de mapas con el que un usuario podía interactuar mediante un navegador y llevar a cabo algunas funciones tales como acercar, alejar, seleccionar capas. A partir de esta iniciativa se desarrollaron aplicaciones similares en los años siguientes. |
| | | | |

| | | | |
|--|---|------|---|
| National Atlas of Canada | Canadian National Atlas Information Service | 1994 | Este mapa interactivo en línea le permitía a los usuarios (ciudadanía en general) seleccionar capas como vías, límites, regiones ecológicas |
| National Geospatial Data Clearinghouse - Alexandria Digital Library | U.S. Geological Survey (USGS) University of California | 1995 | El trabajo conjunto de las dos aplicaciones posibilitaban a los usuarios especificar palabras claves de búsqueda y una ubicación geográfica para obtener un mapa o imágenes de satélite |
| TIGER (Topologically Integrated Geographic Encoding and Referencing) Mapping Service | U.S. Census Bureau | 1995 | Esta aplicación permitía a los usuarios hacer consultas y mostrar sobre un mapa los resultados obtenidos de información demográfica. |
| GRASSLinks (Geographic Resources Analysis Support System) | Susan House - University of California | 1995 | GRASS es un SIG de escritorio que no tenía funcionalidad expuesta en Web. A partir de la implementación de una interfaz se habilitó la comunicación entre navegador, servidor Web, (encargado de recibir las peticiones) y reenviarlo a GRASS para que las procesara. |

| | | | |
|----------|----------|------|---|
| | | | |
| MapQuest | MapQuest | 1996 | Representa el precedente de las aplicaciones actuales, con funcionalidades como búsqueda de locales comerciales y trazado de ruta óptima. |

Etapa III: Web Mapping interactivo, este tipo de aplicaciones ofrecen más opciones de acción entre el usuario y la interfaz del cliente (navegador), hay mayor procesamiento de éste lado y se ofrecen más funcionalidades que en las estáticas.

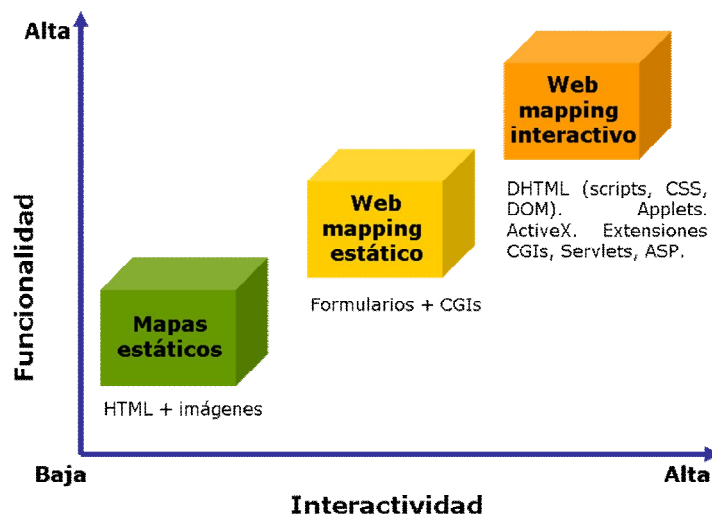


Figura 13. Evolución del Web Mapping. Adaptada de (Peng y Tsou, 2003)

1.3.3 Características de los SIG Web

Internet y la Web permiten acceso instantáneo a la información, sin importar distancia y tiempo, además de estas cualidades, los SIG Web aventajan a los tradicionales o de escritorio por (Fu y Sun, 2011):

- Alcance global: se incrementa la accesibilidad, una vez que la aplicación se libera, el mundo entero puede utilizarla a través de su computador o de dispositivos móviles.

- Gran cantidad de usuarios: los usuarios pueden variar desde decenas hasta cientos, haciendo uso de manera simultánea, lo que requiere de alto desempeño y escalabilidad.
- Multiplataforma: la mayoría de clientes SIG Web son navegadores, los cuales ofrecen diferentes versiones y cumplen con estándares de HTML y JavaScript. Otro tipos de aplicaciones necesitan de la instalación de un ambiente especial para ser ejecutadas (i.e Java, Flex, .NET). La mayor dificultad la presentan los dispositivos móviles, debido a la amplia gama de sistemas operativos y a la incompatibilidad de los browsers.
- Bajo costo, debido al número de usuarios: las organizaciones que requieren de las capacidades de un SIG para su personal, pueden optar por un sistema en Web para ser compartido por todos desde el sitio de trabajo, en el hogar o en campo, lo que reduce los costos de compra de licencias y mantenimiento.
- Facilidad para los usuarios finales: el público al que va dirigido un SIG Web es amplio, ya que abarca desde personas expertas hasta novatos, por lo cual, en el diseño debe predominar la simplicidad, intuición y comodidad
- Actualización unificada: este proceso es más fácil, automático y ahorra tiempo, debido a que los cambios en los programas y datos se realizan en el servidor y se reflejan al instante en los clientes.
- Aplicaciones diversas: la audiencia de los SIG Web demanda variedad de usos, desde formales o sofisticados hasta informales, este último tipo está ganando popularidad en usuarios inexpertos que necesitan emplear técnicas y herramientas de análisis espacial para propósitos personales y comunitarios.

1.3.4 Funcionalidades de un SIG Web

Para Fu y Sun (2011), además de las tareas de captura, almacenamiento, edición, gestión, análisis, compartir y visualizar información espacial, que cumplen los SIG tradicionales, los SIG Web incluyen las siguientes:

- Visualización (mapping) y consultas: son dos funciones comúnmente utilizadas, los datos y los resultados de análisis se presentan en forma de mapas, a la vez que se realizan operaciones de identificación espacial (i.e. el usuario elige un punto y pregunta ¿qué está localizado aquí?) y hacer consultas sobre atributos (i.e. el usuario proporciona parámetros, ¿dónde están las librerías?)
- Colectar información geoespacial: tanto profesionales como aficionados en el campo de los SIG, de manera voluntaria, toman datos en campo, crean y ensamblan información geográfica, que se distribuye de forma gratuita para quien los necesite (i.e. OpenStreetMap). En otras circunstancias, cuando se requiere de fuentes verificadas, de contenido autorizado, que aseguren confiabilidad y exactitud se levantan o se adquieren los datos y se ponen a disposición sólo de los interesados (i.e. sector comercial, gobierno)

- **Diseminar información geoespacial:** la Web es la plataforma por excelencia para distribuir información, los usuarios hacen búsquedas y descargan los datos. Los geoportales son un caso particular de aplicación SIG, en el que se publican recursos geoespaciales, con el objetivo de incentivar la colaboración y cooperación tanto al interior como al exterior de una organización o departamento, ganando eficiencia, reduciendo costos y evitando duplicidad de esfuerzos.
- **Análisis geoespacial:** más allá de la visualización, se pueden brindar funciones que se aplican en la vida diaria, tales como medir distancias / áreas, encontrar ruta óptima, hallar la ubicación según una dirección y análisis de proximidad (i.e. buscar negocios cerca de).

1.3.5 Arquitectura SIG Web y componentes

Fu y Sun (2011), la describen como el resultado de una aplicación Web tradicional más los componentes geográficos. El flujo básico es: el usuario utiliza el SIG Web mediante un cliente -C-, que puede ser un navegador, un programa de escritorio o una aplicación móvil. C envía la petición al servidor Web -SW- a través de Internet vía HTTP. SW reenvía las peticiones al servidor SIG, que retorna los datos necesarios de la base de datos geográfica y procesa la petición, que puede ser generar mapa, realizar una consulta o llevar a cabo una tarea de análisis. Los datos, el mapa, o cualquier resultado se envían al SW y de allí a C, vía HTTP. Finalmente C despliega el resultado al usuario, completando el ciclo de petición - respuesta. La siguiente figura ilustra este proceso:

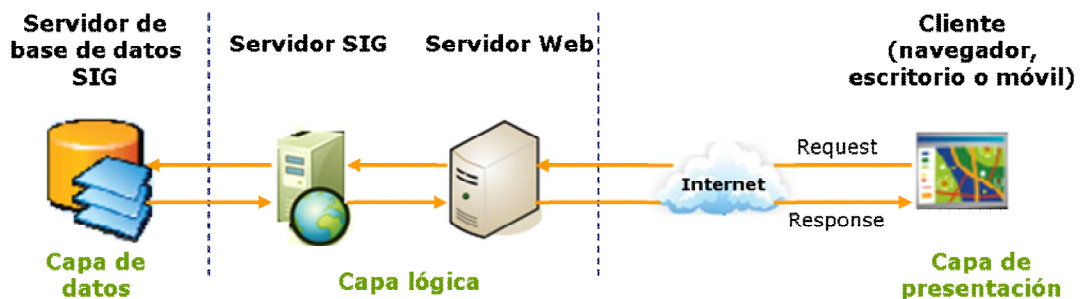


Figura 14. Arquitectura básica SIG Web. Adaptado de (Fu y Su, 2011)

Base de datos SIG: es fundamental en una aplicación SIG Web. La cantidad de respuestas acertadas dadas por el sistema, depende en gran medida de la calidad de la información que contiene. Almacena datos geográficos de varios tipos, tales como vectorial (puntos, líneas y polígonos) y ráster (i.e. imágenes de satélite y aéreas).

Servidor SIG: constituye el núcleo de la arquitectura, su funcionalidad, escalabilidad, desempeño y capacidad de adaptación a las necesidades

particulares son claves para el éxito de una aplicación SIG Web. Proporciona las capacidades de mapping – visualización, análisis de datos espaciales, y la gestión de los servicios publicados.

Servidor Web: encargado de procesar las peticiones http.

Cientes SIG Web: tiene dos roles, el primero es presentar la interfaz de usuario final, con la cual las personas interactúan, colecta entradas, envía peticiones al servidor y presenta los resultados. El segundo, es ejecutar procesamiento espacial, tarea que en su mayoría la realizan clientes pesados. Entre los clientes se encuentran:

- Navegadores Web: emplean tecnologías (i.e AJAX, JavaScript, Flex) con las que se pueden crear interfaces ricas, dinámicas y amigables. Adicionando el uso de APIs, se habilitan funcionalidades SIG.
- Aplicaciones de escritorio: utilizadas para manipulaciones complejas de datos espaciales y tareas de visualización con conexión directa al servidor SIG.
- Dispositivos móviles: se clasifican en dos categorías, los basados en navegador, en las que sus capacidades tienen grandes variaciones y las aplicaciones nativas que pueden tomar ventaja del acceso al dispositivo (i.e tomar posición con GPS)

Arquitectura con cliente liviano y pesado

Arquitectura con cliente liviano: el servidor SIG se encarga de la mayor parte del trabajo, dejándole al cliente solo una mínima responsabilidad. Simplemente envía las peticiones del usuario al servidor, recibe los resultados (ie. imágenes GIF, PNG o JPEG) y los despliega al usuario. Presenta ventajas como: el usuario no necesita instalar software adicional, el navegador es suficiente; ya que el procesamiento se realiza en el servidor, en el cliente no se requiere de computadores de altas prestaciones. Sin embargo, la carga para el servidor es alta y la interacción del usuario es reducida debido a que sólo es HTML plano con funcionalidad JavaScript limitada

Arquitectura con cliente pesado: el cliente desempeña la mayoría de funciones. Usualmente es un navegador con un *plugin* o una aplicación nativa, los cuales se ejecutan localmente. El cliente solicita los datos (i.e. coordenadas, datos en formato vector) al servidor, luego hace el *render* y realiza análisis. Presenta ventajas como: interacción rápida del usuario, debido a que el programa y los datos se encuentran localmente; menor carga del servidor. Aunque, pueden surgir inconvenientes en la instalación de los *plugins* o aplicaciones (i.e. por políticas restrictivas al interior de las organizaciones); también puede resultar insuficiente la capacidad computacional del cliente al realizar sofisticadas operaciones SIG.

La distribución de la carga de trabajo está mediada por la tecnología Web que avanza rápidamente, del lado del cliente los *plugins* y el lenguaje JavaScript se están consolidando y pueden llevar a cabo gran carga de trabajo. Una estrategia de diseño contemporánea distribuye el procesamiento, las prácticas actuales

recomiendan que las funcionalidades más fáciles las realice el *browser* y que las de mayor complejidad recaigan sobre el servidor. La siguiente figura muestra esta concepción:

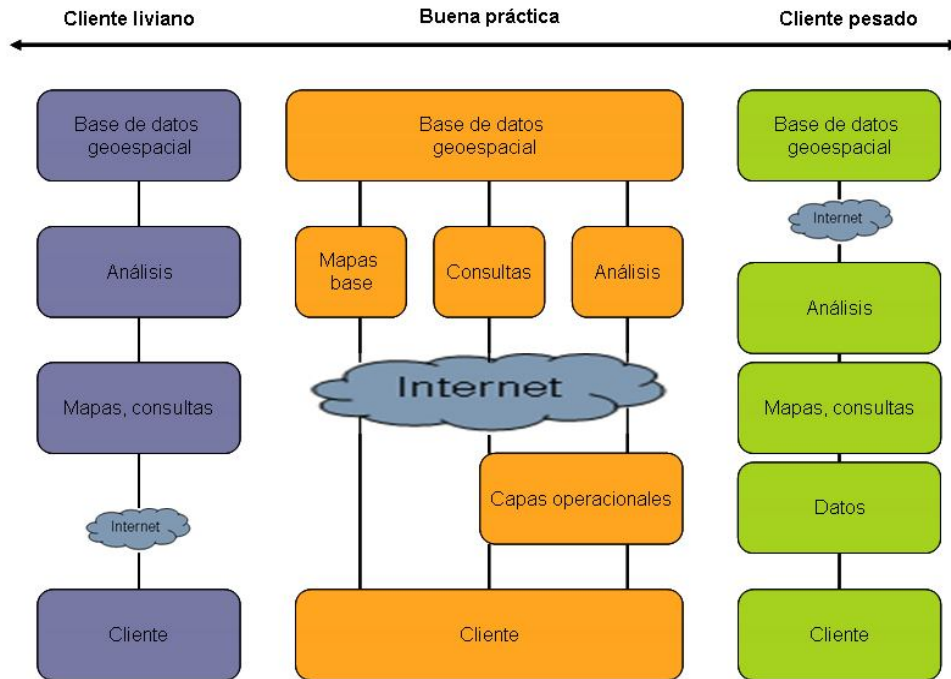


Figura 15. Distribución de la carga de trabajo. Adaptado de (Fu y Sun, 2011)

De acuerdo con Quinn 2008; Brown et al 2008, citado por Fu y Sun 2011, la estrategia se puede resumir en la ecuación:

$$\text{Aplicación SIG Web} = \text{mapas base} + \text{capas operacionales} + \text{herramientas}$$

Donde:

Mapas base: están usualmente pre-creadas o son generadas dinámicamente en el servidor. Proveen la referencia de la localización o el contexto. Tienden a ser estáticas, los cambios no son frecuentes y pueden ser *cacheadas* (creación de determinados grupos de imágenes a ciertas escalas)

Capas operacionales: el *render* es hecho típicamente por el cliente, sin embargo si el tamaño de los datos es muy grande, el servidor es quien asume esta labor. Son dibujadas sobre el mapa base, contiene los temas geoespaciales con los que el usuario final trabaja y visualiza

Herramientas: las tareas fáciles se asignan al cliente y las complejas al servidor SIG. Además de la visualización, se tiene la localización de una dirección o un sitio por su nombre, trazado de rutas, búsqueda de puntos de interés dado ciertos criterios, entre otras.

1.3.6 SIG Web actual

El desarrollo de las aplicaciones SIG Web actuales está marcado por los estándares del Open Geospatial Consortium (OGC) y por el creciente interés que han despertado en las personas, con el consecuente aumento en el número de usuarios. Son dos conceptos que se unen: SIG y Web 2.0.

Una ecuación que resume las características de la Web 2.0 es: (Fu y Sun, 2011)

Web 2.0 = contenido generado por los usuarios + Web como plataforma + experiencia de usuario rica

La fusión del concepto Web 2.0 con SIG implica compartir, comunicar, interoperar, colaborar e integrar información geoespacial.

Como ejemplo de este tipo de aplicaciones está GoogleMaps®, que fue lanzada por Google en 2005, y consiste de un servicio en línea de mapas, con una interfaz de fácil uso y con la capacidad de hacer *mashups*. Dentro de este grupo, también se destacan Google Earth®, Microsoft Bing Maps® y Yahoo Maps®.

Se destaca el concepto de *mashup*, que se refiere a una página Web o una aplicación que dinámicamente combina contenidos o funciones de múltiples sitios Web.

El término *geomashup* está relacionado a un *mashup* en el cual al menos uno de sus contenidos o funciones está georreferenciado. Se integran múltiples fuentes de datos de una localización geográfica común, lo cual permite la superposición de capas de datos, resultando útil para diversidad de análisis (i.e. estudios de riesgo de inundación). Hay dos clases de *mashups*:

- Del lado del servidor SIG, éste envía peticiones a diferentes servicios Web, recibe las respuestas y las une.
- Del lado del navegador, éste envía las peticiones a distintos servicios, recibe las respuestas y despliega la composición de los resultados. Es la forma más común de *mashup*.

La ilustración a continuación representa estos dos tipos:

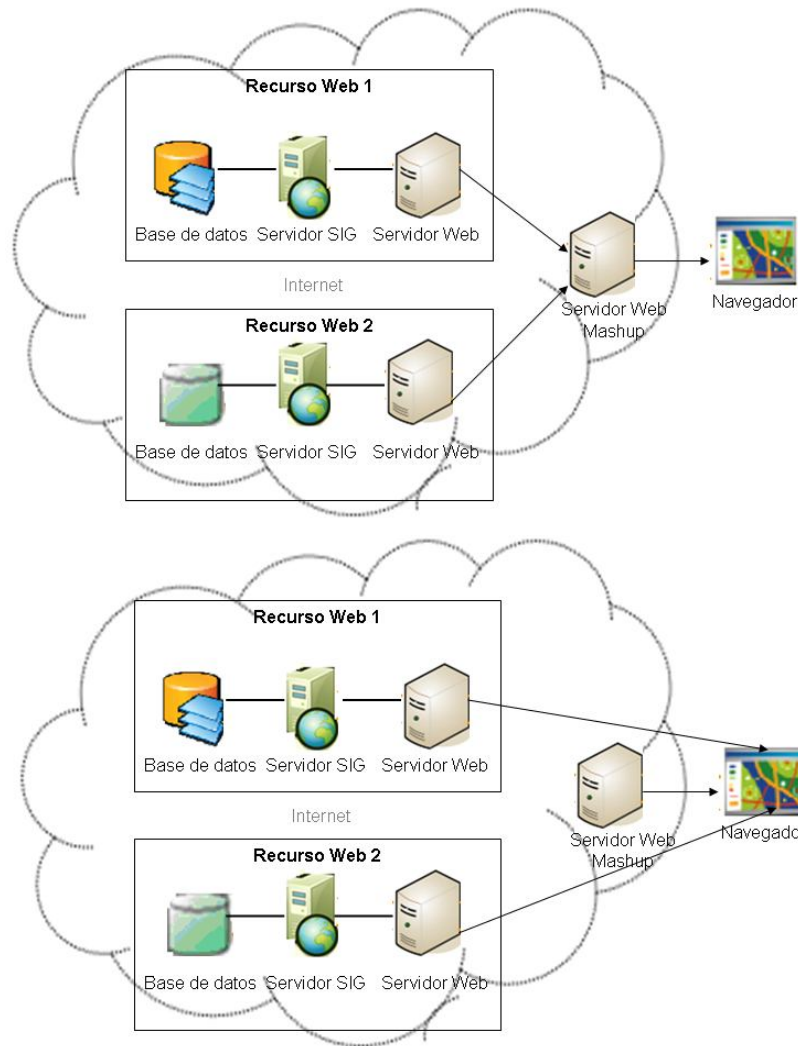


Figura 16. Mashup del lado del servidor (arriba) y del lado del navegador (abajo). Adaptada de (Fu y Sun, 2011)

Los contenidos y las funciones constituyen la base para construir *mashups*, estos recursos pueden ser clasificados en dos grandes grupos:

- Sin APIs: son principalmente HTMLs.
- Con APIs (Application Programming Interfaces), incluyen servicios Web y APIs del lado del navegador.

La siguiente tabla muestra las características de cada uno:

Tabla 7. Servicios Web vs. APIs del lado del navegador. Adaptada de (Fu y Sun, 2011)

| | Servicios Web | APIs del lado del navegador |
|-----------|-----------------|-----------------------------|
| Lenguajes | Puede emplearse | Dependen de un lenguaje |

| | | |
|-------------|---|--|
| | cualquier lenguaje de programación. | específico. Por ejemplo: JavaScript. |
| Capacidades | Proveen funciones del lado del servidor, pueden operar sobre bases de datos. | Permiten el desarrollo de aplicaciones. Comprenden funciones como: interfaces de usuario, interacción con usuario, control del <i>mouse</i> , lógica del lado del navegador. |
| Relaciones | Esperan los llamados de los APIs del lado del navegador o de otros programas. | Encapsulan servicios Web (el llamado y uso es transparente para el usuario). |

El funcionamiento del lado servidor y cliente – navegador se observa en la siguiente figura:

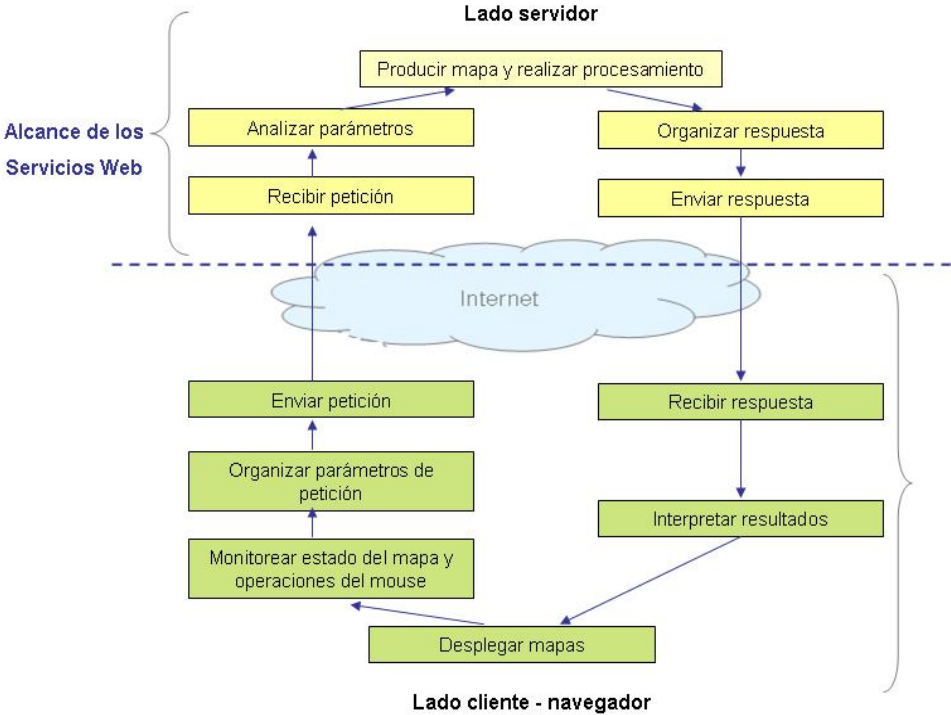


Figura 17. Funcionamiento de APIs del lado del navegador y Servicios Web. Adaptada de (Fu y Sun, 2011)

1.3.7 Experiencia de usuario en aplicaciones SIG Web

Fu y Sun (2011), mencionan la experiencia de usuario como el nivel de satisfacción de una persona al utilizar un producto o servicio. Son tres principios que rigen este aspecto en las aplicaciones SIG Web:

- Velocidad: relacionado con la optimización de los servicios ofrecidos.
- Facilidad de uso: deben ser de manejo intuitivo, tener un enfoque específico y que se autoexpliquen.
- Divertidas: que contengan animaciones, multimedia e interfaces intuitivas, las tecnologías RIA (*Rich Internet Application*), brindan una vivencia similar a las aplicaciones de escritorio.

1.3.8 Mercado y modelos de negocio de los SIG Web.

Fu y Sun (2011), mencionan una forma de evaluar las oportunidades de negocio y modelos de operación, la teoría de *Long Tail* de Anderson (figura 18), en la cual hay muchos usuarios en la cabeza o en la parte superior (mercado de masas), mientras los otros se extienden a lo largo de la curva (nichos de mercado). Aplicado a los SIG Web se observan las siguientes características:

Tabla 8. Mercado y modelos de negocio de los SIG Web. (Fu y Sun, 2011)

| Cabeza | Cola | Mercados potenciales vs. Consolidados | Oportunidades |
|---|---|---|---|
| Tienen funciones comunes pero con limitantes, tales como visualización, búsqueda de puntos de interés, encontrar lugares y direcciones y trazado de rutas. Debido a la gran cantidad de usuarios ofrecen servicios gratuitos y se solventan mediante publicidad. Ejemplos: Google®, Microsoft®, Yahoo®. | Se incluyen instituciones de gobierno, organizaciones y entidades que tienen necesidades especiales o específicas, como datos privados geoespaciales, funciones que involucran a los clientes, toma de decisiones o soporte a los procesos de negocio, entre otros. | Tanto los mercados de la cabeza como los de la cola no están totalmente copados, hay un gran potencial en los dos extremos. Los SIG Web aún se encuentran en etapas iniciales, con gran potencial para explorar y explotar. | Cada vez, la sociedad está más consciente del valor y aplicabilidad de la visualización geoespacial y cuanto más aumente la familiaridad con este tipo de herramientas, mayor será la necesidad de conocimiento en geografía para plantear soluciones en diferentes campos del saber. |

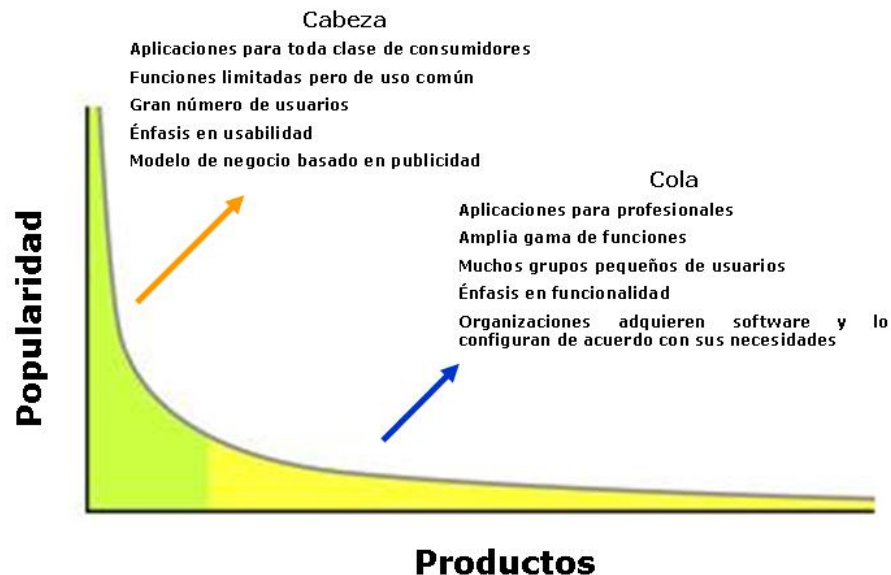


Figura 18. Mercados y modelos de negocio de los SIG Web. Adaptada de (Fu y Sun, 2011)

1.4 SERVICIOS WEB GEOESPACIALES

Las primeras tecnologías SIG Web fueron desarrolladas separadamente y eran vistas como soluciones independientes. Estaban aisladas de otros sistemas y era difícil compartir información y funciones. También, cliente y servidor estaban altamente acoplados, por lo cual los cambios y mejoras implicaban altos costos. Para solucionar estos problemas de interoperabilidad el *Open Geospatial Consortium* (OGC), presentó los estándares de Servicios Web Geoespaciales, basados en los principios de la Arquitectura Orientada a Servicios (SOA), en la cual, las funcionalidades del negocio son empaquetadas como una colección de servicios (o unidades) bajamente acoplados y distribuidos, que pueden ser combinados flexiblemente y reusados para producir nuevas aplicaciones software.

Fu y Sun (2011), expanden la definición de Servicio Web, lo consideran un programa que corre sobre un servidor Web y que expone interfaces a otros programas. Son tres los roles que intervienen: el proveedor (proporciona el servicio), el consumidor (utiliza el servicio) y el registro (es el agente que facilita la colaboración entre proveedor y consumidor). La siguiente figura ilustra el esquema:

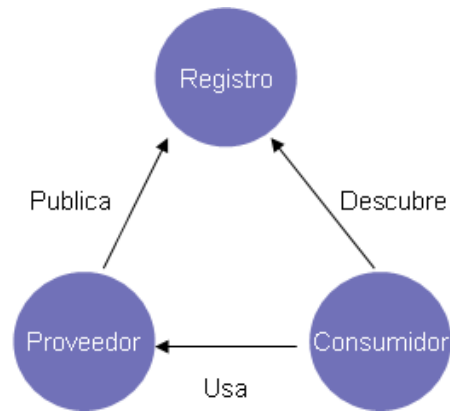


Figura 19. Roles en una arquitectura de Servicios Web. Adaptada de (Fu y Sun, 2011)

Particularmente los servicios Web Geoespaciales se categorizan según el servicio o función que proveen:

- Mapas: es el servicio más común, en el cual los clientes solicitan mapas con un propósito específico y una extensión geográfica determinada, retornando una imagen.
- Datos: permite consultar, editar y sincronizar la información geográfica
- Análisis: desempeña múltiples funcionalidades, por ejemplo, geocodificación (convertir una direcciones a una coordenada), cálculo de rutas, encontrar puntos cercanos a una localización dada, entre otras.
- Catálogo: permite publicar y buscar metadatos de datos espaciales y de otros servicios SIG.

Estándares de Servicios Web Geoespaciales de OGC

El OGC, conformado por representantes de la industria, agencias gubernamentales y universidades, desarrolló una serie de especificaciones, entre las cuales se encuentran:

- *Web Map Service (WMS)*: produce mapas a partir de datos georreferenciados, se genera una imagen, en formatos como PNG, GIF o JPEG. Define tres operaciones: *GetCapabilities*, devuelve los metadatos del nivel de servicio; *GetMap*, retorna un mapa cuyos parámetros geográficos y dimensionales han sido bien definidos; *GetFeatureInfo*, devuelve información de características particulares mostradas en el mapa (opcional). La norma ISO equivalente es la 19128.
- *Web Feature Service (WFS)*: para lectura y escritura de objetos geográficos de formato vectorial. Permite consultar, insertar, actualizar y eliminar objetos geográficos. El estándar define tres operaciones básicas: *GetCapabilities*, devuelve los metadatos del nivel de servicio, tipos de features y operaciones posibles sobre cada tipo de feature; *DescribeFeatureType*, describe la

estructura del tipo de *feature* pedido; *GetFeature*, devuelve el *feature* en formato GML.

- *Web Coverage Service (WCS)*: permite el acceso a coberturas geoespaciales (formato de datos espaciales tipo celda – ráster) que contienen valores o propiedades de las localizaciones geográficas, incluye imágenes de satélite, fotografías aéreas, datos digitales de elevaciones, entre otros. Define tres operaciones: *GetCapabilities*, devuelve los metadatos del nivel de servicio; *DescribeCoverage*, descripción detallada de una o varias coberturas; *GetCoverage*, obtiene una cobertura o parte de ella.
- *Catalog Service for the Web (CSW)*: soporta la publicación y búsqueda de metadatos geoespaciales. Proporciona mecanismos para clasificar, registrar, describir, buscar, mantener y acceder a información acerca de los recursos disponibles en la red, que pueden ser datos o servicios.

La arquitectura de servicios Web del OGC, permite disponer de múltiples servicios de información geográfica con diferentes finalidades (visualización, descarga, geoprocésamiento, localización) a través de la red. Esta capacidad es posible debido a las reglas establecidas para que dichos servicios hagan pública sus funcionalidades y la forma en que se debe interactuar (como se deben enviar las peticiones de servicio) vía métodos y protocolos abiertos y estandarizados. La siguiente figura muestra una configuración de la arquitectura:

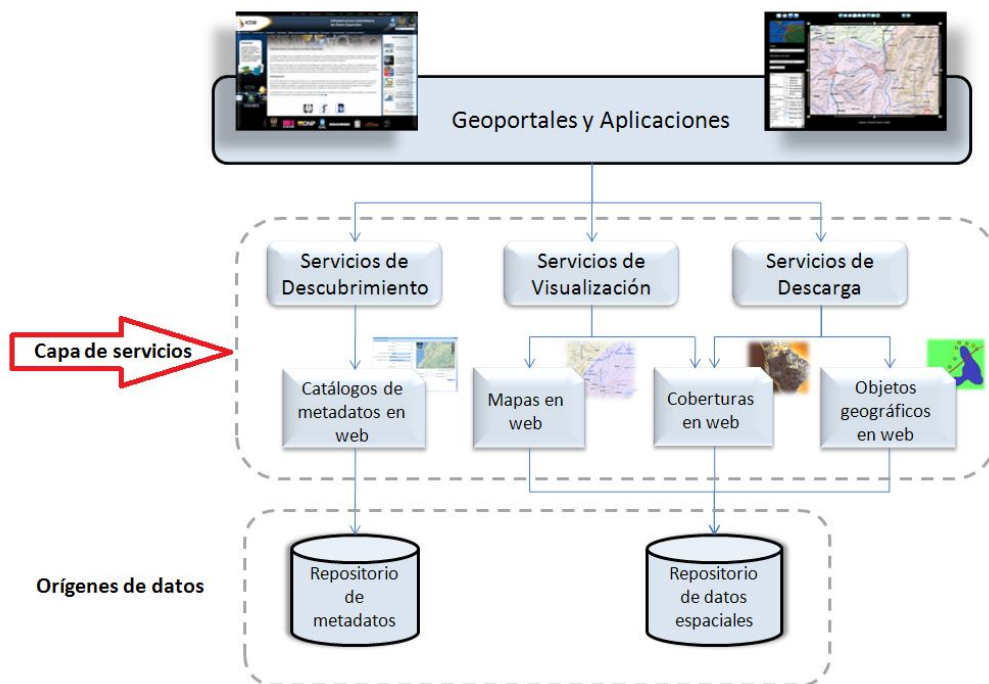


Figura 20. Arquitectura de servicios Web del OGC. Adaptada de http://www.icde.org.co/web/guest/geoservicios_icde

1.5 TECNOLOGÍAS PARA LA CONSTRUCCIÓN DE APLICACIONES SIG WEB

Las aplicaciones informáticas empleadas en la construcción de un SIG Web son diversas, están en constante evolución, se relacionan unas con otras para implementar una arquitectura funcional y provienen tanto del mundo del software libre como del propietario.

1.5.1 Software libre y propietario

Free and Open Source Software (FOSS) es software liberado bajo licencia que asegura a los usuarios la libertad de uso (para cualquier propósito), modificación y redistribución. El acceso al código fuente es pre requisito. Su utilización no genera cargos por derechos o regalías. La siguiente tabla resume tres conceptos importantes:

Tabla 9. Comparación de los conceptos Free software, Open source software y software propietario. Elaboración propia

| Free software | Open source software | Software propietario |
|---|--|--|
| <p>Es una visión ideológica, el creador del término, Richard Stallman la empleó para referirse a que el código fuente debe ser libre y a la no comercialización de la industria del software.</p> <p>Se apoya en la <i>Free Software Foundation</i> (FSF)</p> | <p>El término fue ideado por Eric Raymond y otros compañeros, con una visión más pragmática, enfocada hacia la forma de adoptar comercialmente el FOSS o tipos de licencia y al proceso de desarrollo software utilizado. Se soporta en la <i>Open Source Initiative</i> (OSI)</p> | <p>Quien lo desarrolla tiene los derechos exclusivos sobre el código fuente y los usuarios solo tienen acceso a la versión en binario.</p> <p>Tiene un precio de compra que otorga una licencia, pero que limita la redistribución y cambios</p> |

1.5.2 Tipos de licencias para productos FOSS

Tabla 10. Licencias FOSS. Elaboración propia

| Categoría de licencia | Descripción | Tipos | Productos |
|-----------------------|--|-----------------------|---|
| | <p>También se denominan virales y apoyan la noción sobre la total libertad del software. La más representativa es la GNU Public Licence (GPL), que ampara el concepto de copyleft, que implica que cualquier</p> | <p>GPL, LGPL, OSL</p> | <p>GNU/Linux Operating System es GPL.</p> |

| | | | |
|-------------|---|--------------------------|---------------------------|
| Recíproca | programa que incluya código de este tipo, también debe ser liberado bajo esta licencia. | | |
| Académica | Tiene pocas restricciones, el principal requisito es reconocer el trabajo a quienes contribuyeron previamente | Apache licence, BSD, MIT | El core Apple OS X es BSD |
| Corporativa | Permite la combinación de FOSS y software propietario, las compañías tienen el control sobre las licencias de los productos derivados | MPL, Qt Public Licence | Mozilla Firefox es MPL |

La siguiente figura muestra la relación entre diferentes tipos de licenciamiento FOSS:

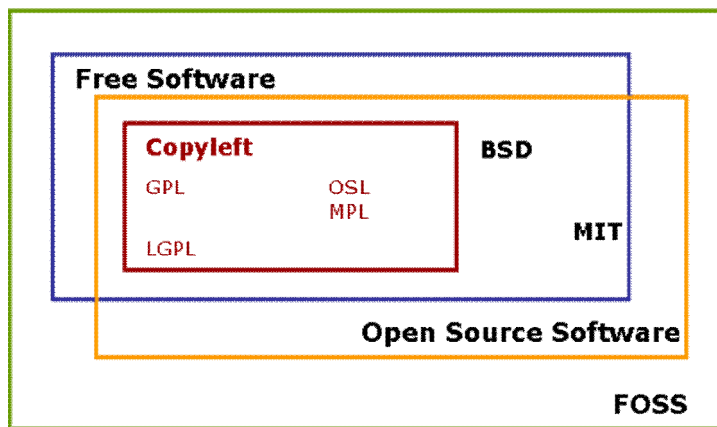


Figura 21. Free Software and Open Source Software. Adaptado de diagrama de Chao-Kuei <http://www.fsf.org/licensing/essays/categories.html>

1.5.3 Software libre para SIG Web: FOSS4GIS y OSGeo

Como el nombre lo indica, FOSS4GIS es software SIG liberado bajo una licencia FOSS. La comunidad que desarrolla, usa y promueve el FOSS4GIS ha crecido notablemente en los últimos años, se encuentra integrada por organizaciones comerciales, instituciones gubernamentales e iniciativas privadas que contribuyen monetariamente, con habilidades personales y código fuente. El núcleo sobre el que gira una amplia variedad de proyectos es la fundación internacional llamada "*The Open Source Geospatial Foundation*" - OSGeo, que se encarga de brindar apoyo financiero, organizacional y legal, además de velar por la alta calidad de los productos desarrollados.

1.5.4 Herramientas software para construcción de aplicaciones SIG Web

A continuación se presentan las fichas técnicas de algunas herramientas software libres y propietarias, clasificadas de acuerdo con la función que cumplen dentro de la arquitectura: clientes, servidores y bases de datos.

Rol: Clientes

Tabla 11. Características de OpenLayers. Elaboración propia

| | |
|---|--|
| Nombre: OpenLayers | |
| Licencia: FreeBSD | |
| Lenguaje programación: JavaScript | Versión estable y fecha de lanzamiento: 2.12. Julio 2012 |
| Organización promotora: Metacarta | Sitio Web: http://openlayers.org/ |
| Descripción general: biblioteca de funciones que permite integrar mapas dinámicos, de distintas fuentes de información, en una página Web. Tiene similitud al API de Google Maps. No tiene dependencia de la parte del servidor. | Año de lanzamiento: 2005 |
| Documentación para desarrollador: presentaciones, ejemplos en vivo, tutoriales, descripción del API. Idioma: inglés y algunos recursos en otros idiomas | |

Tabla 12. Características de GeoExt. Elaboración propia.

| | |
|--|--|
| Nombre: GeoExt | |
| Licencia: puede ser distribuido bajo BSD. Pero al ser dependiente de <i>ExtJS</i> y dependiendo del tipo de aplicación puede pasar a ser de tipo comercial. | |
| Lenguaje programación: JavaScript | Versión estable y fecha de lanzamiento: 1.1. Diciembre 2011 |

| | | |
|--|------------------------------------|-------------|
| Organización promotora: OpenGeo | Sitio http://geoext.org/ | Web: |
| Descripción general: es una librería que combina OpenLayers con el framework <i>ExtJS</i> . Provee un conjunto de <i>widgets</i> personalizables para visualización, edición y estilo de datos geoespaciales. | Año de lanzamiento: 2009 | |
| Documentación para desarrollador: tiene tutorial básico, códigos de ejemplo, ejemplos funcionales y la referencia al API .Idioma: inglés | | |

Rol: Servidores geográficos

Tabla 13. Características de Geoserver. Elaboración propia

| | | |
|--|---|-------------|
| Nombre: Geoserver | | |
| Licencia: GNU General Public License (GPL) version 2 | | |
| Lenguaje programación: Java | Versión estable y fecha de lanzamiento: 2.2. Septiembre 2012 | |
| Organización promotora: Refrations Research Inc, The Open Planning Project, Axios | Sitio http://geoserver.org | Web: |
| Descripción general: es un Servidor SIG Web que permite publicar mapas y datos provenientes de diferentes fuentes. Soporta los estándares: WMS, WFS transaccional, WCS, Filter Encoding, SLD, GML. Sistemas operativos: Windows, Linux, Mac | Año de lanzamiento: 2003 | |
| Documentación para desarrollador: dado que es un servidor, listo para su uso, la documentación incluye instalación y manejo de la interfaz de gestión. Idioma: inglés. También tienen disponible documentación para quienes deseen contribuir con el proyecto, presenta los requisitos para su instalación, forma de obtener código fuente, configuración entorno de desarrollo, guía de programación, realización de pruebas | | |

Tabla 14. Características de Mapserver. Elaboración propia

| | |
|--|---|
| Nombre: MapServer | |
| Licencia: MIT-style license | |
| Lenguaje programación: C/C++ | Versión estable y fecha de lanzamiento: 6.0.3. Mayo 2012 |
| Organización promotora: Metacarta | Sitio Web: http://mapserver.org/ |
| Descripción general: es un programa tipo CGI. Este motor se encarga de hacer el <i>render</i> de los datos geográficos. Despliega y consulta múltiples formatos vectoriales, ráster y bases de datos. Maneja proyecciones cartográficas al vuelo. Entre los estándares soportados están: WMS, WFS no transaccional y WCS. Sistemas operativos: Linux, Windows, Mac OS X, Solaris. | Año de lanzamiento: mediados años 90. |
| Documentación para desarrollador: completa, expone proceso de instalación, configuración, múltiples ejemplos. Idioma: inglés. | |

Framework completo: Cliente - Servidor:

Tabla 15. Características de Geomajas. Elaboración propia

| | |
|---|--|
| Nombre: Geomajas | |
| Licencia: GNU Affero general public licence (AGPL) v3. Tiene también licencia comercial | |
| Lenguaje programación: Java y JavaScript | Versión estable y fecha de lanzamiento: 1.11 distribution 1.12.36. Septiembre 2012 |

| | |
|---|---|
| Organización promotora: DFC Software Engineering; GeoSparc | Sitio Web: http://www.geomajas.org |
| <p>Descripción general: Es un framework para el desarrollo de visualizadores avanzados en aplicaciones empresariales.</p> <p>Brinda una plataforma de integración de datos geoespaciales, del lado servidor, permitiendo a múltiples usuarios el control y gestión de datos desde un navegador.</p> <p>Se centra en las funcionalidades de la parte servidora, manteniendo el lado de cliente realmente liviano.</p> | Año de lanzamiento: 2008 |
| <p>Documentación para desarrollador: ilustra la arquitectura, el API, la manera de configurar el entorno de desarrollo. Idioma: inglés.</p> <p>Tiene disponible documentación para quienes deseen contribuir con el proyecto.</p> | |

Tabla 16. Características de Mapbender. Elaboración propia

| | |
|---|---|
| Nombre: Mapbender | |
| Licencia: GNU GPL y Simplified BSD license | |
| Lenguaje programación: Del lado servidor: PHP. Del lado cliente: Javascript (jQuery y jQuery UI) | Versión estable y fecha de lanzamiento: 2.7.3. Julio 2012 |
| Organización promotora: CCGIS | Sitio Web: http://www.mapbender.org/ |
| Descripción general: es un entorno para la publicación de Geoportales, y para el registro, visualización, navegación, monitorización y manejo de niveles de acceso seguros a | Año de lanzamiento: 2001 |

| | |
|---|--|
| servicios de Infraestructuras de Datos Espaciales. | |
| <p>Documentación para desarrollador: dado que es un framework enfocado al usuario final, los tutoriales que se encuentran son sobre instalación, configuración y utilización. Idiomas: inglés y alemán. Tiene disponible documentación para quienes deseen contribuir con el proyecto.</p> | |

Tabla 17. Características de MapFish. Elaboración propia

| | |
|---|--|
| <p>Nombre: Mapfish</p> | |
| <p>Licencia: BSD</p> | |
| <p>Lenguaje programación: basado en Pylons, framework Python</p> | <p>Versión estable y fecha de lanzamiento: 2.2 Junio 2011</p> |
| <p>Organización promotora: Camptocamp</p> | <p>Sitio Web: http://mapfish.org/</p> |
| <p>Descripción general: provee herramientas específicas para la creación de servicios Web que permiten consultar y editar elementos geográficos. También proporciona un toolbox JavaScript orientado a la creación de RIAs, basado en ExtJS, OpenLayers y GeoExt</p> | <p>Fecha primera versión: 2009</p> |
| <p>Documentación para desarrollador: es básica, explica a nivel general la instalación, creación de servicios, uso del toolbox Javascript y conexión a bases de datos espaciales. Idioma: inglés</p> | |

Tabla 18. Características de ArcGIS Server. Elaboración propia

| | |
|---|--|
| Nombre: ArcGIS Server | |
| Software licenciado: el costo y la funcionalidad ofrecida dependen del nivel (<i>basic, standard y advanced</i>). El API JavaScript es de descarga gratuita, pero está protegido por copyright. | |
| Lenguaje programación: del lado del cliente: JavaScript, Flex y framework microsoft Silverlight. | Versión estable y fecha de lanzamiento: 10.1. Febrero 2012. |
| Organización promotora: ESRI | Sitio Web: http://www.esri.com/software/arcgis/arcgisserver |
| Descripción general: soporta diferentes bases de datos (DB2, Informix, SQL Server, Oracle, PostgreSQL, Netezza). Los servicios Web que implementa son REST, SOAP y OGC. Puede ser desplegado físicamente o virtualizarse en infraestructuras <i>cloud</i> . Ofrece visores listos para su uso. Tiene tres APIs para desarrollo del lado del cliente: JavaScript, Flex y Microsoft Silverlight; también proporciona herramientas para creación de aplicaciones móviles. Sistemas operativos: Windows y algunas distribuciones de Linux. | Fecha primera versión: 2007 (ArcGIS Server 9.2) |
| Documentación para desarrollador: los APIs para desarrollo del lado del navegador son completas, se explica su uso mediante ejemplo, piezas de código, ejemplos en vivo. Idioma: inglés. | |

Rol: Bases de datos geográficas

Tabla 19. Características de PostgrsSQL- Postgis. Elaboración propia

| | |
|---|---|
| Nombre: PostgrsSQL- Postgis | |
| Licencia: GNU / GPL | |
| Lenguaje programación: SQL, pSQL | Versión estable y fecha de lanzamiento: 2.0.1. Junio 2012 |
| Organización promotora: Refraction Research | Sitio Web: http://postgis.refractor.net/ |
| Descripción general: es el soporte para almacenar, consultar y manipular datos geoespaciales de la base de datos relacional PostgreSQL. Algunas funciones de geoprocetamiento con las que cuenta son: áreas de influencia, uniones, <i>overlays</i> , distancia. Compatible con estándar OGC <i>Simple Features for SQL</i> . Sistemas operativos: Windows/GNU/Linux/Mac OS X. | Año de lanzamiento: 2001 |
| Documentación para desarrollador: se encuentran manuales, presentaciones, ejercicios y datos para practicar el manejo de la base de datos, tanto en el sitio de la compañía que lo promueve como de terceros. | |

Tabla 20. Características de Oracle Locator. Elaboración propia

| | |
|---|---|
| Nombre: Oracle Locator | |
| Licencia: dentro de los productos que ofrece Oracle, está Express Edition (XE), la cual es gratuita, recomendada para instituciones educativas y estudiantes para la realización de ejercicios académicos. | |
| Lenguaje programación: | Versión estable y fecha de lanzamiento: 11g release 2. |

| | |
|---|--|
| Organización promotora: Oracle | Sitio http://www.oracle.com/index.html Web: |
| Descripción general: es una característica de todas las ediciones. Provee los tipos de datos, operadores y capacidades de indexación de Oracle Spatial más un conjunto limitado de subprogramas (funciones y procedimientos). Proporciona conjunto de funciones y procedimientos para almacenar, gestionar, consultar y realizar análisis espacial. Incluido sin costo adicional en: Oracle XE, Oracle Standard Edition, Oracle Standard Edition One, Oracle Enterprise Edition. | Año de lanzamiento: 2000 |
| Documentación para desarrollador: completa, organizada, explica funcionalidades, brinda ejemplos para uso, idioma: inglés. | |

1.5.5 Patrones de codificación

Es de vital importancia para el desarrollo de una aplicación, sea tradicional Web o SIG Web, comprender y saber emplear patrones, ya que facilitan la organización del código y su reutilización.

En general, un patrón es una solución (o una aproximación) probada y reusable a un tipo de problema.

En esta sección se encuentran dos patrones que ayudan a la organización del código y mejora de performance: separación de comportamiento y *namespaces*, los cuales están orientados a JavaScript, lenguaje ampliamente utilizado para el desarrollo de aplicaciones SIG Web.

Para aclarar conceptos (i.e. *closures*) se recomienda estudiar el anexo B y otros textos sobre el lenguaje JavaScript como el de Stefanov, (2008).

Separación de comportamiento

Los tres bloques constitutivos de una página Web son:

- Contenido (HTML): debe utilizarse la cantidad adecuada de etiquetas, de manera que sean suficientes para describir la semántica. Idealmente debe estar libre de elementos de formato visual, por ejemplo: el atributo *style* debe evitarse, la etiqueta `` no debe emplearse.
- Presentación (CSS): son las hojas de estilo, encargadas de los formatos visuales.

- Comportamiento (JavaScript): debe ser adicionado, preferiblemente en archivos externos y claramente diferenciado del contenido y la presentación. Se recomienda: minimizar el número de etiquetas *<script>*, evitar el uso entre líneas de HTML de eventos (*onclick*, *onmouseover*, reemplazar por *addEventListener*, *attachEvent*), no utilizar expresiones de CSS, insertar en el *<body>* de la página un solo archivo .js

Ejemplo:

Tabla 21. Ejemplo de separación de comportamiento. Tomado de (Stefanov, 2008)

| HTML | JavaScript |
|---|--|
| <pre> <body> <form id="myform" method="post" action="server.php"> <fieldset> <legend>Search</legend> <input name="search" id="search" type="text" /> <input type="submit"/> </fieldset> </form> <script type="text/javascript" src="behaviors.js"> </script> </body> </pre> | <pre> myevent.addListener('myform', 'submit', function(e){ e= myevent.getEvent(e); myevent.stopPropagation(e); var el = document.getElementById('search'); if(!el.value){ myevent.preventDefault(e); alert('Favor ingresar cadena de búsqueda '); } }); </pre> |

Namespaces

Esta es una vía para minimizar el número de variables globales: se crea un solo objeto global y todas las demás variables y funciones se convierten en propiedades de él. Ejemplo:

```
var MYAPP = MYAPP || {}; // namespace global
```

Se tiene una propiedad *event* del objeto MYAPP:

```
MYAPP.event = {} // es un sub-objeto
```

Se adicionan los métodos de una forma sencilla:

```
MYAPP.event = {  
    addListener:function(el,type,fn){  
        //implementar funcionalidad...  
    },  
    removeListener:function(el,type,fn){  
        // implementar funcionalidad...  
    },  
    getEvent:function(e){  
        // implementar funcionalidad...  
    }  
    //otros métodos o propiedades  
};
```

Propiedades y métodos privados

Los lenguajes clásicos tienen los siguientes modificadores:

Public: todos los usuarios de un objeto pueden acceder a sus propiedades (o métodos).

Private: únicamente el objeto tiene acceso a sus propiedades.

Protected: solo los objetos que hereden del objeto en cuestión pueden tener acceso a las propiedades.

JavaScript no tiene una sintaxis especial para denotar propiedades de tipo privado, pero se pueden utilizar variables locales y métodos dentro del constructor para lograr cierto nivel de protección. Ejemplo: se muestra como *Button* puede hacer uso de propiedades locales privadas, en esta implementación *styles* y

`setStyles` son privados, el constructor las emplea internamente, pero no están disponibles fuera de la función.

```
var MYAPP = { };

MYAPP.dom.Button = function (text,conf){

    var styles={

        font: `Verdana`,

        border: `1px solid black`,

        color: `black`,

        background: `grey`

    };

    function setStyles(){

        for(var i in styles){

            b.style[i]=conf[i] || styles [i];

        }

    }

    conf = conf || {};

    var b = document.createElement( `input` );

    b.type = conf[ `type` ] || `submit`;

    b.value=text;

    setStyles();

    return b;

};
```

Funciones privadas como métodos públicos

En este caso se asigna la función privada a una propiedad disponible públicamente. Se ilustra el concepto con el ejemplo a continuación, en el que se definen `_setStyle` y `_getStyle` como funciones privadas, pero se asignan a las públicas `setStyle` y `getStyle`.

```
var MYAPP = { };

MYAPP.dom = function (){

    var _setStyle=function(el,prop,value){
```

```

        console.log('setStyle ');
    };
    var _getStyle=function(el,prop){
        console.log('getStyle ');
    };
    return{
        setStyle: _setStyle,
        getStyle:_getStyle,
        yetAnother: _setStyle
    };
}) ();

```

Si se llama a *MYAPP.dom.setStyle()*, se invocará a la función privada *_setStyle()*. También se puede sobrescribir *setStyle()* desde el exterior:

```
MYAPP.dom.setStyle()= function(){alert ('b ')};
```

Funciones que se autoejecutan

Este es un patrón que ayuda a mantener el namespace limpio, envolviendo el código en una función anónima y ejecutándola inmediatamente. De esta forma cualquier variable de la función es local (mientras se use *var*) y se destruyen cuando la función retorna, si no hacen parte de un *closure*. Es especialmente adecuado para tareas de inicialización que se realizan cuando se carga el script. El patrón puede extenderse para crear y retornar objetos. Si la creación es compleja e involucra alguna inicialización, se puede hacer en la primera parte de la función autoejecutable y retornar un solo objeto, el cual puede acceder a cualquier propiedad privada de las declaradas en el principio. Ejemplo:

```

var MYAPP = { };
MYAPP.dom = function (){
    // código de inicialización ...
    function _private(){
        //...implementar funcionalidad
    }
    return{
        getStyle: function(el,prop){

```



```

        console.log('getStyle ');
        _private();
    },
    setStyle: function(el,prop,value){
        console.log('setStyle');
    }
};
} ();

```

1.5.6 Patrones de Diseño

A continuación se presenta un conjunto de patrones representativos.

Singleton

Es un patrón creacional, es útil para asegurar que sólo hay un objeto de una clase dada. En JavaScript, es el patrón por defecto y más natural, ya que no hay clases. Cada objeto es único. Si no se copia y no se usa como prototipo de otro, se mantiene como el único objeto de su tipo. La más básica implementación es:

```
var single ={};
```

Ejemplo de implementación de un singleton utilizando sintaxis de tipo clase:

```

var my_log = new Logger();
my_log.log('some event');
// ... otro código complementario...
var other_log = new Logger();
other_log.log('some event');
alert(other_log===my_log.log); // true

```

Aunque se emplee *new*, solo una instancia se crea y ésta se retorna en las llamadas consecutivas.

Otra forma de implementar es utilizar una variable global para almacenar la única instancia. El código del constructor puede ser:

```

function Logger(){
    if(typeof global_log===undefined){
        global_log=this;
    }
}

```

```
    }  
    return global_log;  
}
```

Cuando se llama al constructor, se tiene el siguiente resultado:

```
var a = new Logger();  
var b = new Logger();  
alert (a === b); // true
```

La desventaja está en el uso de una variable global, que puede ser sobrescrita en cualquier momento (aún accidentalmente) y se perdería la instancia.

En la alternativa que se presenta a continuación, se asigna la única instancia a una propiedad del constructor:

```
function Logger(){  
    if(typeof Logger.single_instance === 'undefined'){  
        Logger.single_instance=this;  
    }  
    return Logger.single_instance;  
}
```

Cuando, `var a = new Logger()`, `a` apuntará a la propiedad recién creada `Logger.single_instance`. Una llamada subsecuente, `var b = new Logger()`, tendrá como resultado que `b` apunte a la misma `Logger.single_instance`.

Esta forma resuelve el problema de namespace global, ya que no se crean variables globales, el único inconveniente es que la propiedad del `Logger` es visible públicamente, entonces puede ser sobrescrita, corriendo el riesgo que la instancia se pierda o se modifique.

Entonces para evitar la dificultad de la sobrescritura, se recurre a una propiedad privada, protegiéndola mediante un *closure*.

Factory

Es otro patrón creacional, es útil cuando se tienen tipos de objetos similares y no se sabe de antemano cual se usará. Basado en las entradas del usuario u otro tipo de criterio se determina al vuelo qué clase de objeto se necesita. El código ejemplo ilustra el patrón, en el cual hay tres constructores diferentes, que

implementan funcionalidades parecidas, se toma la URL y puede crear un nodo DOM, un enlace o una imagen:

```
var MYAPP = { };  
  
MYAPP.dom = {};  
  
MYAPP.dom.Text = function(){  
    this.insert=function(where){  
        var txt = document.createTextNode(this.url);  
        where.appendChild(txt);  
    };  
};  
  
MYAPP.dom.Link = function(){  
    this.insert=function(where){  
        var link = document.createElement('a');  
        link.href=this.url;  
        link.appendChild(document.createTextNode(this.url));  
        where.appendChild(link);  
    };  
};  
  
MYAPP.dom.Image = function(){  
    this.insert=function(where){  
        var im = document.createElement('img');  
        img.src=this.url;  
        where.appendChild(im);  
    };  
};
```

La forma de utilizar los constructores es la misma, se establece la *url* y luego se llama al método *insert()*;

```
var o = new MYAPP.dom.Image();  
o.url = 'http://www.dominio.com/images/logo.png';
```

```
o.insert(document.body);
```

En caso de que no se conozca con anterioridad qué tipo de objeto se requiere, por ejemplo cuando el usuario decide en tiempo de ejecución por medio de un click en un botón, probablemente se emplearían las sentencias *if* o un *switch*:

```
var o;  
  
if(type === 'Image'){  
    o= new MYAPP.dom.Image();  
}  
  
if(type === 'Link'){  
    o= new MYAPP.dom.Link();  
}  
  
if(type === 'Text'){  
    o= new MYAPP.dom.Text();  
}  
  
o.url = 'http://...';  
  
o.insert();
```

Esta solución trabaja bien, pero si se tienen varios constructores, el código se extiende demasiado. Además si se está creando una librería o un *framework*, no se tiene conocimiento de los nombres exactos de las funciones con anterioridad. Aquí es cuando el patrón demuestra su utilidad, determinando dinámicamente el tipo:

```
MYAPP.dom.factory=function(type){  
    return new MYAPP.dom[type];  
}
```

Luego se reemplazan los *if* con:

```
var o = MYAPP.dom.factory(type);  
  
o.url = 'http://...';  
  
o.insert();
```

El método *factory()* en este caso es simple, en un escenario real se validarían los valores de *type* y opcionalmente se establecería una tarea común para los objetos.

Decorator

Es un patrón estructural, en el cual se tiene un objeto base y un conjunto de objetos decoradores que le proveen funcionalidades extra. El siguiente código muestra un ejemplo sencillo:

```
var obj = {  
    function: doSomething(){  
        console.log('sure', 'asap');  
    },  
    //...  
};  
  
obj = obj.getDecorator('deco1');  
obj = obj.getDecorator('deco13');  
obj = obj.getDecorator('deco5');  
obj.doSomething();
```

Se inicia con un simple objeto que tiene el método *doSomething()*, luego se eligen algunos decoradores (identificados por el nombre), cada uno de los cuales tiene su implementación de *doSomething()*, que primero llamará al mismo método del decorador previo y luego ejecutará el propio. Cada vez que se adiciona un decorador, se sobrescribe la base *obj* con una versión mejorada. Cuando se finaliza el proceso de agregar decoradores, al llamar a *doSomething()*, se ejecutan secuencialmente los métodos de cada uno.

Observer

Este patrón es de tipo comportamental, maneja la forma en que interactúan y se comunican entre sí los objetos. Se diferencian dos partes:

- Uno o más objetos publicadores: los cuales anuncian que ha sucedido algo importante.
- Uno o más subscriptores: que están sintonizados con uno o varios publicadores, escuchan sus anuncios y actúan adecuadamente.

Hay dos subtipos de patrones Observer:

- Push: los publicadores son responsables de notificar a cada subscritor.
- Pull: los subscriptores monitorean los cambios de estado de los publicadores.

El siguiente ejemplo es una implementación del modelo push, el objeto *observer* tiene las siguientes propiedades y métodos:

- Un array de *subscribers* que son solo funciones callback
- *addSubscriber()* y *removeSubscriber()* para adicionar y remover del array.
- *publish()*, que toma los datos y llama a todos los subscriptores para entregárselos.
- *make()* toma cualquier objeto y lo convierte en publicador, adicionándole todo los métodos pertinentes.

```

var observer= {
    addSubscriber: function(callback){
        this.subscribers[this.subscribers.length]=callback;
    },
    removeSubscriber: function(callback){
        for (var i=0; i< this.subscribers.length;i++){
            if(this.subscribers[i]===callback ){
                delete(this.subscribers[i]);
            }
        }
    },
    publish: function(what){
        for (var i=0; i< this.subscribers.length;i++){
            if(typeof this.subscribers[i]=== 'function '){
                this.subscribers[i](what);
            }
        }
    },
    make: function(o){
        for (var i in this){
            o[i]=this[i];
            o.subscribers=[];
        }
    }
}

```

```
};
```

Ahora se crean algunos publicadores, el objeto *blogger* llama a *publish* cada vez que un nuevo post de un blog está listo:

```
var blogger= {  
  writeBlogPost:function(){  
    var content = 'Today is' + new Date();  
    this.publish(content);  
  }  
};
```

Para convertir a *blogger* en publicador:

```
observer.make(blogger);
```

Ahora con dos simples objetos denominados *jack* y *jill*:

```
var jack= {  
  read: function(what){  
    console.log('I just read that '+ what);  
  }  
};  
  
var jill= {  
  gossip: function(what){  
    console.log('You didn\'t hear it from me, but '+ what);  
  }  
};
```

jack y *jill* pueden subscribirse al objeto *blogger*, implementando los callbacks que quieran ser llamados cuando algo es publicado:

```
blogger.addSubscriber(jack.read);
```

```
blogger.addSubscriber(jill.gossip);
```

Lo que sucede cuando *blogger* escribe un nuevo post, es que *jack* y *jill* son notificados:

```
blogger.writeBlogPost();
```

El resultado será: I just read that Today is Sun Apr 06... y You didn't hear it from me, but Today is Sun Apr 06...

Module

Se utiliza para emular el concepto de clases, de manera que un objeto tenga variables y métodos (públicos o privados). Se encapsula un módulo para evitar conflictos con otros. Un ejemplo en código:

```
var myNamespace = (function () {  
    var myPrivateVar, myPrivateMethod;  
    myPrivateVar = 0; // contador, variable privada  
    myPrivateMethod = function( foo ) {  
        console.log( foo );  
    };  
    return {  
        myPublicVar: "foo",  
        // fx pública que utiliza var privadas  
        myPublicFunction: function( bar ) {  
            myPrivateVar++;  
            myPrivateMethod( bar );  
        }  
    };  
})();
```

Revealing Module

Es una versión mejorada del Module, en el cual se definen todas las funciones y variables de alcance privado y se retorna un objeto anónimo que apunta a las funcionalidades privadas que se quiera revelar como públicas. A continuación un ejemplo de implementación:

```
var myRevealingModule = function () {  
    var privateCounter = 0;  
    function privateFunction() {  
        privateCounter++;  
    }  
}
```



```

    }

    function publicFunction() {
        publicIncrement();
    }

    function publicIncrement() {
        privateFunction();
    }

    function publicGetCount(){
        return privateCounter;
    }

    // Presentar punteros públicos a
    // funciones y propiedades privadas
    return {
        start: publicFunction,
        increment: publicIncrement,
        count: publicGetCount
    };
}());

myRevealingModule.start();

```

1.6 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Es importante tener en cuenta las metodologías, ya que la creación de una aplicación exige definir un proceso lógico que conduzca desde el levantamiento de requisitos hasta la puesta en producción. En este apartado se trata la tradicional o RUP y dos del tipo ágil.

1.6.1 Metodología tradicional. Rational Unified Process (RUP)

RUP es una metodología desarrollada por Rational Software, su actual propietarios es IBM. RUP se caracteriza por ser una metodología general, es decir, extensa y contiene una gran colección de procesos, artefactos y roles que

deben ser adaptados dependiendo del tipo de proyecto a abordar, hace énfasis en la documentación.

RUP se caracteriza por: (Rational, 1998)

- Proceso dirigido por Casos de Uso, además de ser una herramienta para especificar los requisitos del sistema también guían su diseño, implementación y prueba, es decir, se constituyen en un elemento integrador y una guía de trabajo que no sólo inicia el proceso de desarrollo sino que proporciona un hilo conductor, permitiendo establecer continuidad entre los artefactos generados en las diferentes actividades del proceso de desarrollo.
- Proceso centrado en la arquitectura, se presta especial atención en el establecimiento temprano de la arquitectura buscando que esta no se vea impactada por cambios posteriores durante la construcción y el mantenimiento del sistema.
- Proceso iterativo e incremental, busca dividir el desarrollo en una secuencia de iteraciones que permiten revisar al final de cada una, si existen nuevos requisitos y entregar continuamente aplicaciones funcionales al cliente, incluyendo toda la documentación de la aplicación como manual de usuario e instalación y demás documentos que apliquen en la entrega.
- Arquitectura basada en componentes: desarrollo temprano de una arquitectura que aunque robusta permita acomodarse a los cambios, entendible intuitivamente y que promueva el reuso.
- Modelado visual: utilizando UML (*Unified Modeling Language*).
- Verificación continua de la calidad: el aseguramiento se realiza durante todo el proceso, en todas las actividades, se involucran todos los participantes, empleando métricas y criterios objetivos.
- Gestión de los cambios: describe cómo controlar, hacer seguimiento y monitorear los cambios (i.e. en modelos, códigos, documentos).

Un proyecto realizado siguiendo RUP se divide en cuatro fases: inicio, elaboración, construcción y transición. En cada fase se ejecutan una o varias iteraciones y se componen de nueve actividades: modelado de negocio, análisis de requisitos, análisis y diseño, implementación, pruebas, distribución, gestión de la configuración, gestión del proyecto, gestión del entorno. La imagen a continuación representa lo anterior:

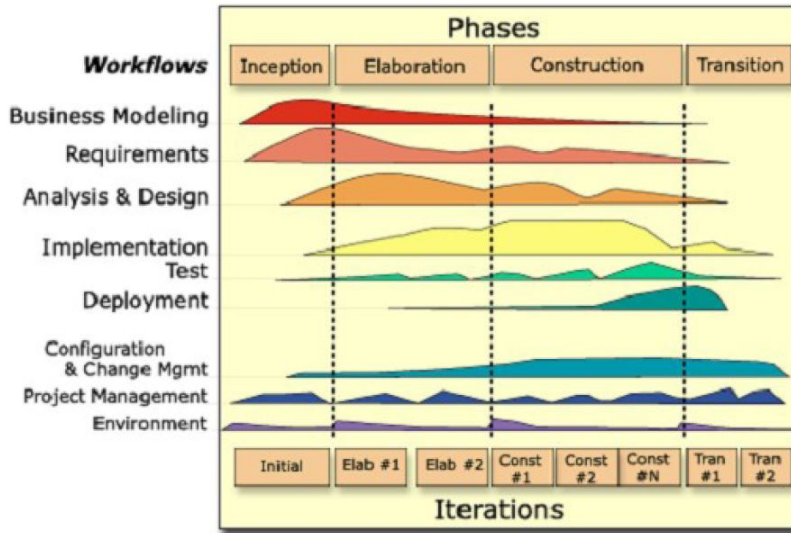


Figura 22. Estructura de proceso RUP. Tomada de (Gaona, 2011)

El siguiente gráfico muestra el flujo de trabajo y los artefactos generados en RUP:

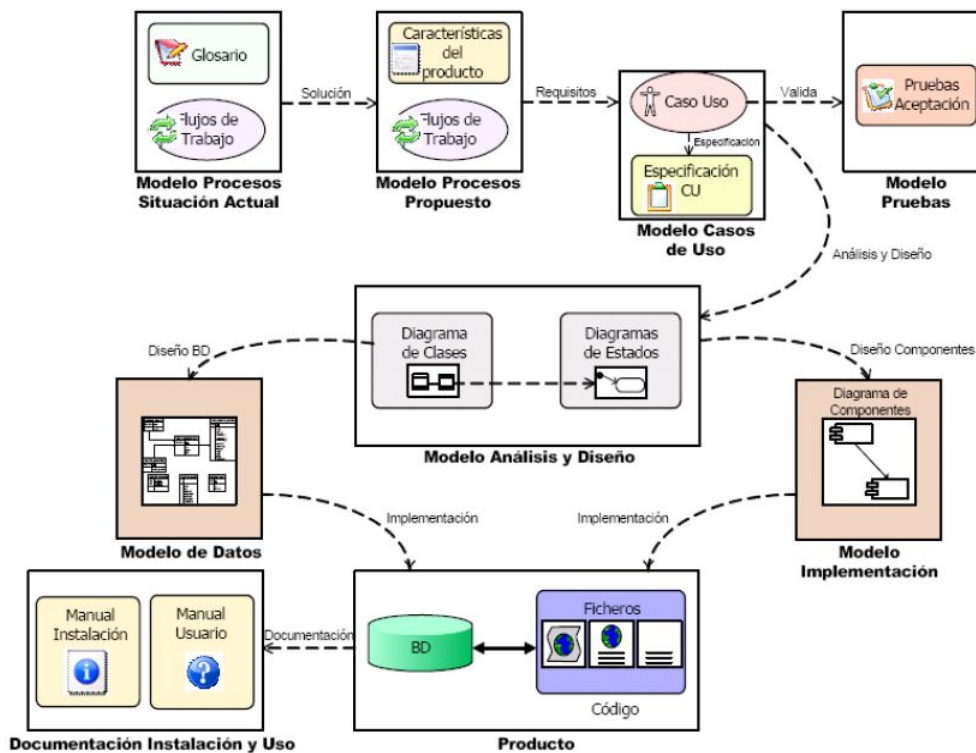


Figura 23. Flujo de trabajo y artefactos RUP. Tomada de (Gaona, 2011)

1.6.2 Metodologías ágiles

Existe gran variedad de metodologías de desarrollo ágiles, para la construcción del marco teórico, se seleccionaron XP y Scrum debido a su representatividad e influencia en la ingeniería de software.

eXtreme Programming (XP)

XP está conformado por un conjunto de prácticas de desarrollo software, empaquetadas por Kent Beck y Ward Cunningham. Sus raíces están en la comunidad orientada a objetos, especialmente en los programadores SmallTalk.

Runeson y Greberg (2004), señalan en su artículo que XP es una metodología de desarrollo liviana, cuyo énfasis está en el grupo de trabajo, la comunicación, la retroalimentación, la simplicidad y la solución de problemas. Está enfocada al desarrollo más que a una dirección técnica, por esta razón no se hace documentación excesiva sobre los avances, las entregas, los requisitos, etc.

XP está construido sobre cuatro valores: Comunicación, Retroalimentación, Simplicidad y Coraje/Valor. A través de la comunicación dentro y fuera del proyecto se asegura que el producto correcto es desarrollado; una retroalimentación rápida y frecuente provee habilidades para corregir la dirección del proyecto, la simplicidad significa construir el producto correcto y no un producto para posibles necesidades futuras; y el coraje/valor es necesario para mantener la franqueza y la comunicación.

XP se caracteriza porque es un proceso iterativo donde el cliente se convierte en un integrante más del grupo de trabajo, la programación se hace en parejas que se rotan frecuentemente para un mayor conocimiento del sistema, se hacen entregas frecuentes y en muy corto plazo, está dirigido por historias de usuario escritas por el cliente (es la forma de capturar requisitos) y se tiene la firme convicción de desarrollar sólo lo que se necesita para cuando se necesita; pero lo más importante es que al estar enfocado en el desarrollo, en XP se escriben las pruebas antes de iniciar con el código y las entregas hechas deben pasar al 100% dichas pruebas.

Scrum

Es una de las metodologías ágiles más difundidas, no está concebida como método independiente, sino que se promueve como complemento de otras metodologías como XP, o RUP, ya que se enfoca en valores y prácticas de gestión, sin pronunciarse sobre requisitos, implementación y demás cuestiones técnicas.

Scrum se define como un proceso de gestión y control que implementa técnicas de control de procesos, asume que el proceso de desarrollo del sistema es complicado e impredecible por lo que sólo puede ser descrito a grandes rasgos y lo define como un conjunto holgado de actividades, por esta razón implementa controles para manejar los procesos y riesgos inherentes (Schwaber, 1997).

Algunas características de Scrum son:

- Equipos autodirigidos y autoorganizados. En Scrum no existe ningún personaje que se encargue de dirigir a todo el equipo.
- Una vez escogida la carga de trabajo no se le hacen adiciones, en caso de que se le agregue algo se recomienda quitar otra tarea que la equilibre nuevamente.
- Encuentros diarios, en mesa redonda, con el fin de hacer seguimiento y disminuir cualquier retraso en el avance del proyecto.
- Proceso iterativo en la etapa de desarrollo con iteraciones de una a cuatro semanas de duración.
- Al inicio de cada iteración se realiza planeamiento adaptativo con el cliente.
- Al final de cada iteración se hace demostración del producto a los participantes externos o clientes y se hacen revisiones para discutir los cambios que deben hacerse y asumir los riesgos a que haya lugar.

En Scrum el equipo total no debe sobrepasar los 10 integrantes, se recomienda siete más o menos dos, sin embargo, si el equipo es muy grande se puede dividir en varios subequipos del tamaño recomendado y se define un equipo central que se encarga de la coordinación de todos.

Resumen de Metodologías Ágiles y Tradicionales

La siguiente tabla resume una comparación entre las tres metodologías consignadas:

Tabla 22. Comparación RU, XP y Scrum. Tomada de (Gaona, 2011)

| Característica | RUP | XP | Scrum |
|-------------------------------------|-----|----|-------|
| Documentación estricta | SI | NO | NO |
| Proyectos geográficamente dispersos | SI | NO | NO |
| Realización de iteraciones | SI | SI | SI |
| Procesos estrictamente sistemáticos | SI | NO | NO |
| Resultados rápidos | NO | SI | SI |
| Utilización de artefactos | SI | SI | SI |

2. CAPÍTULO 2

CONSTRUCCIÓN DE APLICACIONES PILOTO

Dado que el objetivo de la investigación, es generar una serie de lineamientos para la construcción de una aplicación SIG Web, este capítulo recoge el proceso de desarrollo de prototipos, los cuales constituyen la base experimental para su posterior formulación.

2.1 CARACTERÍSTICAS DE LA APLICACIÓN PILOTO

El enfoque del proyecto está en el desarrollo de una aplicación del lado del cliente, que se accede mediante un navegador, por esta razón y para facilitar la labor, se optó por aprovechar los datos levantados por el Centro IG - laboratorio de SIG, del campus EAFIT sede Medellín (Geodatabase y *shapefiles*). Se tomaron algunos requisitos de la aplicación Campus Interactivo (<http://webapps.eafit.edu.co/mapaInteractivo/campusEAFIT.jsf>) para definir el *backlog* de los prototipos a realizar. La documentación del proceso de desarrollo bajo metodología RUP puede consultarse en el documento de Valencia (2010).

Tabla 23. Backlog de los prototipos

| # | Requisito | Tipo | Implicaciones | Prioridad |
|---|--|--------------|--|-----------|
| 1 | Localizar oficina o aula | Funcional | | Alta |
| 2 | Localizar sitio de interés | Funcional | | Alta |
| 3 | Desplegar información complementaria cuando se realicen 1y 2. | Funcional | Debe existir archivos con este tipo de información (i.e fotografías) | Media |
| 4 | Debe tener herramientas de zoom in/out, pan, reload y <i>select by point</i> (identificar) | Funcional | La herramienta <i>select by point</i> brindará sólo información de algunas capas. | Alta |
| 5 | La imagen de fondo que tenga el mapa puede ser de un servicio externo. | Funcional | La fuente puede ser Google, OpenStreetMap, Bing o ArcGIS | Baja |
| 6 | La base de datos puede ser Oracle 10g. | No funcional | Debe tener habilitada su extensión <i>locator</i> , para que pueda manejar la información geográfica | Media |

2.2 SELECCIÓN DE HERRAMIENTAS SOFTWARE

La selección de herramientas para la construcción de una aplicación SIG Web no es un proceso trivial, existe un ecosistema amplio tanto de soluciones libres como propietarias, cuya configuración ofrece también varias posibilidades, gracias a los estándares de interoperabilidad del OpenGeospatial Consortium. Por ejemplo, la siguiente figura muestra la diversidad de clientes open source que pueden utilizarse:

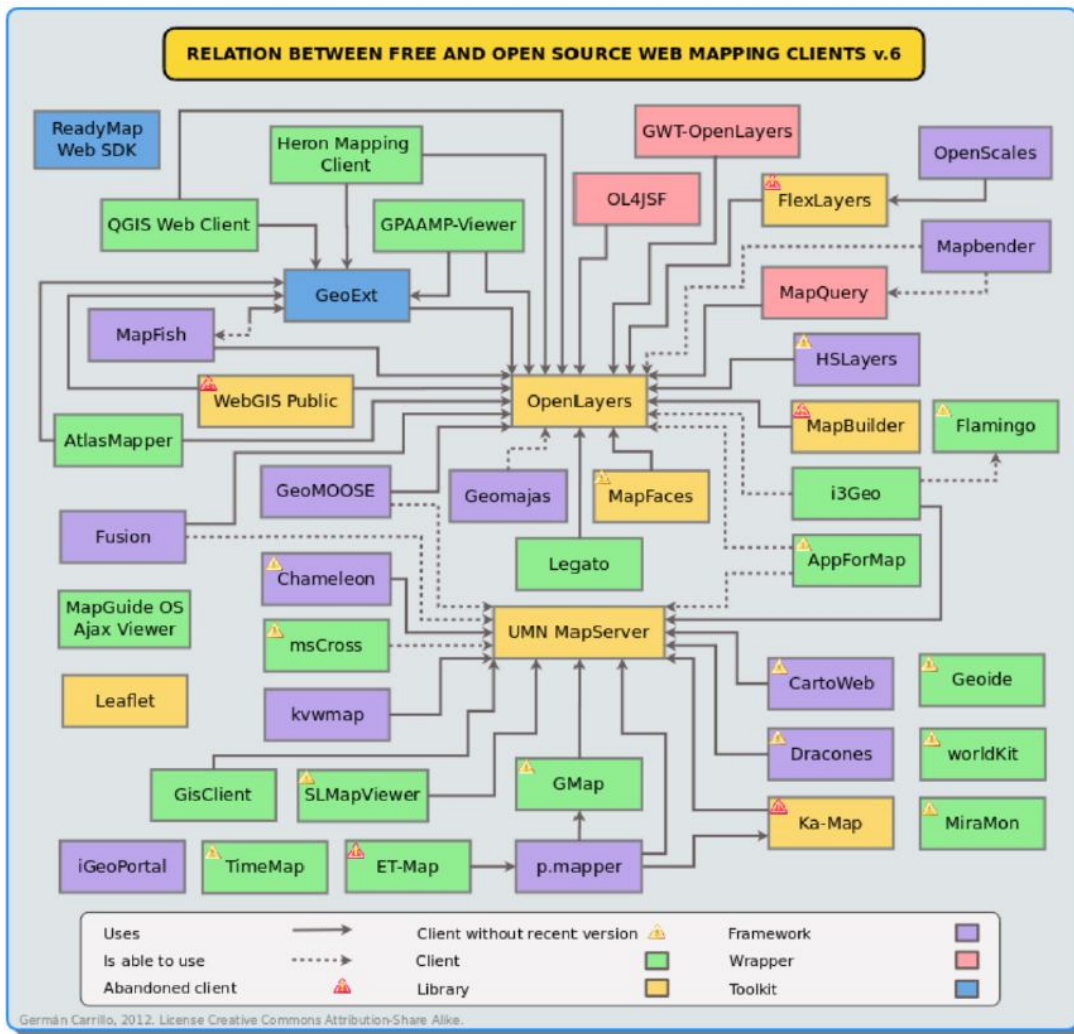


Figura 24. Clientes para aplicaciones de Web Mapping. Tomada de (Carrillo, 2012)

Para el desarrollo de la investigación se realizó una selección de herramientas. En primera instancia se dividió en dos conjuntos: libres y propietarios. En la categoría de software propietario, se decidió por el producto ArcGIS Server de ESRI, ya que es una de las casas fabricantes más reconocidas a nivel mundial, utilizado ampliamente tanto por entidades gubernamentales como por empresas privadas y EAFIT cuenta con la licencia para su uso, en la versión 9.3 (actualizada bajo contrato).

De otra parte, para el software libre, los criterios que se tuvieron para realizar el filtro de selección fueron:

- Hacer parte de los proyectos cobijados por el Open Source GeoSpatial Foundation, ya que al estar bajo la sombrilla del OSGeo tienen, en líneas generales, mayor estabilidad y perdurabilidad, que aquellos que se sostienen por sí mismos, ya que cuentan con beneficios como acceso a infraestructura (i.e. repositorio para código y documentación), respaldo de marca (*branding*) - visibilidad, patrocinio y una organización más formal. Además se incentiva la colaboración entre los proyectos de la fundación. También OSGeo, tiene un capítulo de la comunidad hispanohablante, que se caracteriza por su proactividad y por brindar acompañamiento mediante lista de correo y wiki principalmente. El listado de herramientas de interés para la presente investigación está en la tabla a continuación:

Tabla 24. Proyectos de clientes ligeros y Servidores que hacen parte de OSGEO

| Clientes ligeros o de navegador Web | Servidores |
|---|---|
| OpenLayers - Cliente GIS de Navegador Web Geomajas - Cliente GIS de Navegador Web Mapbender - Framework de Geoportal MapFish - Framework de Web Mapping GeoMoose - Portal web SIG | GeoServer MapServer deegree QGIS Server - Servicio WMS |

- El lenguaje de programación debe figurar en el ranking de Tiobe, los resultados al mes de junio de 2011 y octubre de 2012, se muestran en la siguiente tabla, se señala la estabilidad de JavaScript.

| Position Oct 2012 | Position Oct 2011 | Delta in Position | Programming Language | Ratings Oct 2012 | Delta Oct 2011 | Status | Position Jun 2011 | Position Jun 2010 | Delta in Position | Programming Language | Ratings Jun 2011 | Delta Jun 2010 | Status |
|-------------------|-------------------|-------------------|----------------------|------------------|----------------|--------|-------------------|-------------------|-------------------|----------------------|------------------|----------------|--------|
| 1 | 2 | ↑ | C | 19.822% | +2.11% | A | 1 | 2 | ↑ | Java | 18.580% | +0.82% | A |
| 2 | 1 | ↓ | Java | 17.193% | -0.72% | A | 2 | 1 | ↓ | C | 16.278% | -1.91% | A |
| 3 | 6 | ↑↑↑ | Objective-C | 9.477% | +3.23% | A | 3 | 3 | = | C++ | 9.830% | -0.55% | A |
| 4 | 3 | ↓ | C++ | 9.260% | +0.19% | A | 4 | 6 | ↑↑ | C# | 6.844% | +2.06% | A |
| 5 | 5 | = | C# | 6.530% | -0.19% | A | 5 | 4 | ↓ | PHP | 6.602% | -2.47% | A |
| 6 | 4 | ↓↓ | PHP | 5.669% | -1.15% | A | 6 | 5 | ↓ | (Visual) Basic | 4.727% | -0.93% | A |
| 7 | 7 | = | (Visual) Basic | 5.120% | +0.57% | A | 7 | 10 | ↑↑↑ | Objective-C | 4.437% | +2.07% | A |
| 8 | 8 | = | Python | 3.895% | -0.05% | A | 8 | 7 | ↓ | Python | 3.899% | -0.20% | A |
| 9 | 9 | = | Perl | 2.126% | -0.31% | A | 9 | 8 | ↓ | Perl | 2.312% | -0.97% | A |
| 10 | 11 | ↑ | Ruby | 1.802% | +0.28% | A | 10 | 20 | ↑↑↑↑↑↑↑ | Lua | 2.039% | +1.55% | A |
| 11 | 10 | ↓ | JavaScript | 1.261% | -0.93% | A | 11 | 12 | ↑ | JavaScript | 1.501% | -0.58% | A |
| 12 | 12 | = | Delphi/Object Pascal | 1.097% | -0.01% | A | 12 | 11 | ↓ | Ruby | 1.484% | -0.61% | A |
| 13 | 13 | = | Lisp | 0.947% | -0.08% | A | 13 | 9 | ↓↓↓ | Delphi/Object Pascal | 1.070% | -1.50% | A |
| 14 | 18 | ↑↑↑↑ | Pascal | 0.839% | +0.12% | A | 14 | 16 | ↑↑ | Lisp | 0.935% | +0.28% | A |
| 15 | 15 | ↑ | Lua | 0.728% | -0.07% | A | 15 | 15 | = | Pascal | 0.731% | +0.00% | A |
| 16 | 20 | ↑↑↑↑ | Ada | 0.654% | +0.04% | B | 16 | - | = | Assembly* | 0.673% | - | B |
| 17 | 15 | ↓↓ | PL/SQL | 0.630% | -0.27% | B | 17 | 21 | ↑↑↑↑ | Transact-SQL | 0.651% | +0.16% | B |
| 18 | 25 | ↑↑↑↑↑↑ | Visual Basic .NET | 0.599% | +0.12% | A- | 18 | 25 | ↑↑↑↑↑↑ | RPG (OS/400) | 0.637% | +0.22% | B |
| 19 | 21 | ↑↑ | MATLAB | 0.591% | +0.02% | B | 19 | 23 | ↑↑↑↑ | Ada | 0.606% | +0.17% | B |
| 20 | 19 | ↓ | Assembly | 0.568% | -0.05% | B | 20 | - | = | Scheme* | 0.579% | - | B |

Figura 25. Ranking de los 20 lenguajes de programación utilizados más frecuentemente, de acuerdo a Tiobe para junio de 2011 y octubre de 2012

- Fecha de última versión inferior a 1 año.
- Experiencia en el manejo de la herramienta
- Pertinencia: en el caso de los clientes, que sean APIs que permitan programar y no tipo Geoportal que son orientados hacia usuario final, que solo los configura. Para los servidores que tengan las funcionalidades suficientes y necesarias para el desarrollo de los pilotos.

También se tuvieron en cuenta las siguientes cifras que corresponden a proyectos o casos de estudio presentados durante los eventos FOSS4GIS y las Jornadas de SIG libre de la Universidad de Girona de los años 2009 y 2010

Tabla 25. Proyectos y casos de estudio presentados los eventos Jornadas SIG libre y FOSS4GIS entre 2009 y 2010. Elaboración propia

| Herramienta | III Jornadas SIG libre 2009 | IV Jornadas SIG libre 2010 | FOSS4GIS 2009 | FOSS4GIS 2010 |
|-------------|-----------------------------|----------------------------|---------------|---------------|
| MapServer | 6 | 5 | 10 | 9 |
| GeoServer | 2 | 7 | 5 | 5 |
| MapBender | 2 | - | - | 1 |
| OpenLayers | 6 | 6 | 12 | 11 |
| MapFish | - | 2 | 3 | 2 |

Otro parámetro, aunque no crítico, fueron cursos ofrecidos, gratuitos y con costo en idioma español, en instituciones gubernamentales y Universidades entre años 2010 -2011, ya que esto señala la adopción por parte de la comunidad académica y la necesidad del público por aprender este tipo de tecnologías:

Tabla 26. Cursos ofrecidos de herramientas de SIG libre, en español entre 2010 y 2011. Elaboración propia

| Institución | Temática |
|---|---|
| IGN Instituto Geográfico Nacional España | MapServer, GeoServer, OpenLayers |
| Universidad de Girona – SIGTE | OpenLayers |
| Universidad Cantabria | OpenLayers |
| Universidad Da Coruña - ETS de Caminos, Canales y Puertos | OpenLayers |
| Instituto del Mar del Perú – Universidad de Sevilla | OpenLayers, MapServer |
| IGAC Instituto Geográfico Agustín Codazzi | Geoserver (Capacitaciones sobre Infraestructura de Datos Espaciales - Geoservicios) |
| Universidad Politécnica Madrid – Instituto Geográfico Nacional España (IGN) | Geoserver (Capacitaciones sobre Infraestructura de Datos Espaciales) |
| Universidad de la República Uruguay | OpenLayers, MapServer en SIG Empresariales. |

Al listado de herramientas libres acogidas por OSGeo (tabla 24), se aplicaron los criterios de selección, los resultados se muestran a continuación:

Tabla 27. Criterios evaluados para selección de herramientas. Elaboración propia

| Criterio | Herramientas | | | | | | | | |
|--|--------------|------------|-------------|-----------|------------|-------------|-------------|-------------------------------|--|
| | 1.OL | 2.GeoMajas | 3.MapBender | 4.MapFish | 5.GeoMoose | 6.GeoServer | 7.MapServer | 8.deegree | 9.QGISServer |
| Lenguaje programación en top 20 de Tiobe | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fecha última versión inferior a 1 año | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Experiencia en el manejo de la herramienta | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Pertinencia | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ (Orientado hacia IDEs *) | ✗ (Sólo WMS y trabaja mejor unido a QGIS) |
| Número creciente de proyectos presentados en FOSS4GIS y Jornadas SIG Libre - SIGTE | ✓ | ? | ✗ | ? | ? | ✓ | ✗ | ? | ? |
| Instituciones / Universidades ofrecen cursos de capacitación en español. | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ? | ✗ |
| Total de ✓ | 6 | 3 | 3 | 3 | 2 | 6 | 5 | 2 | 2 |

✓ = SI ✗ = NO ? = No se registran datos o es indeterminado

* IDE: Infraestructura de datos espaciales.

Después de analizar las anteriores posibilidades y cumpliendo con los criterios establecidos, se optó en la parte cliente por OpenLayers, herramienta que cuenta con una madurez de más de 10 años. Sin embargo se necesitaba de un *framework* que soportara la creación de interfaces de usuarios (IU) y que tuviera componentes (*widgets*) para desplegar información geoespacial, por esto se eligió a GeoExt.

Para el servidor, se escogió Geoserver, también se tomó en consideración la comparación de Graser (2010), entre Geoserver y Mapserver, que se resume en la siguiente tabla:

Tabla 28. Geoserver vs Mapserver. Tomado de Graser (2010)

| | Geoserver | Mapserver |
|----------------|---|--|
| WMS | Los dos son buenos | Quizá un poco mejor |
| WFS | Es mejor, soporta WFS-T | no WFS-T |
| Tecnología | JEE | CGI |
| Administración | Herramienta Web | La generación del <i>mapfile</i> puede apoyarse en QGIS, pero no se compara con la herramienta de Geoserver. |
| Extensibilidad | Buena para desarrolladores Java | Mediante PHP Mapscript, buena para desarrolladores PHP |
| Cartografía | Usa SLDs estándares. | Potente; los estilos hacen parte del <i>mapfile</i> . |
| Servicios | Un servicio WMS/WFS/WCS para todos los usuarios | Un <i>mapfile</i> se traduce a un servicio |
| Consultas | Filtros CQL y OGC. | Sentencias SQL embebidas |

2.3 DESARROLLO DE APLICACIÓN PILOTO CON SOFTWARE LIBRE

En la siguiente figura y tabla se ilustra la arquitectura software y se resume el *stack* de herramientas seleccionadas:

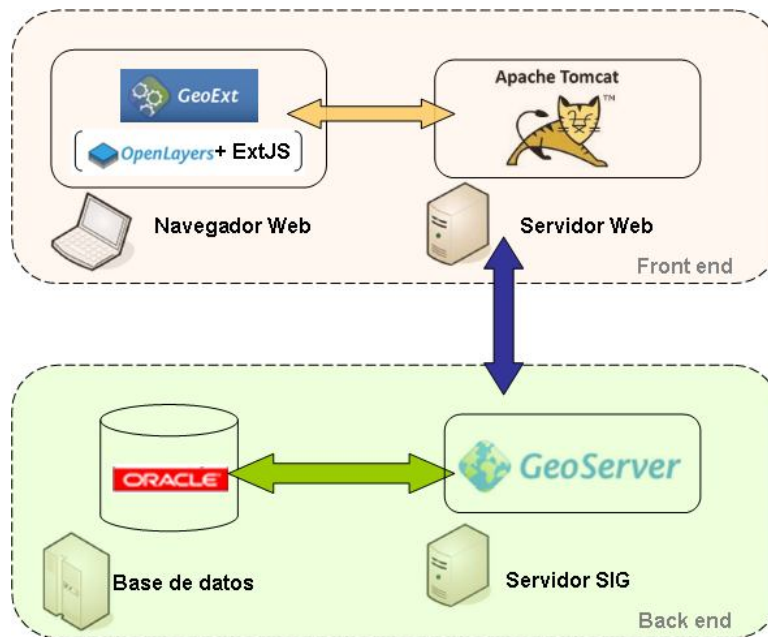


Figura 26. Arquitectura de aplicación con software libre

Tabla 29. Stack de herramientas para construir aplicación con software libre

| | |
|-------------------|--|
| Sistema Operativo | Ubuntu 12.04. Linux 3.2.0 |
| Base de datos | Oracle XE 10g. |
| Servidor de mapas | Geoserver, versión 2.2.1 |
| Servidor Web | Apache Tomcat 7 |
| Cliente | OpenLayers v 2.10 GeoExt v 1.0. Basado en framework ExtJS v3.2 |

Siguiendo el orden de la arquitectura presentada, primero se describe el proceso de construcción de la base de datos, luego la publicación en un servidor y finalmente la construcción de la aplicación:

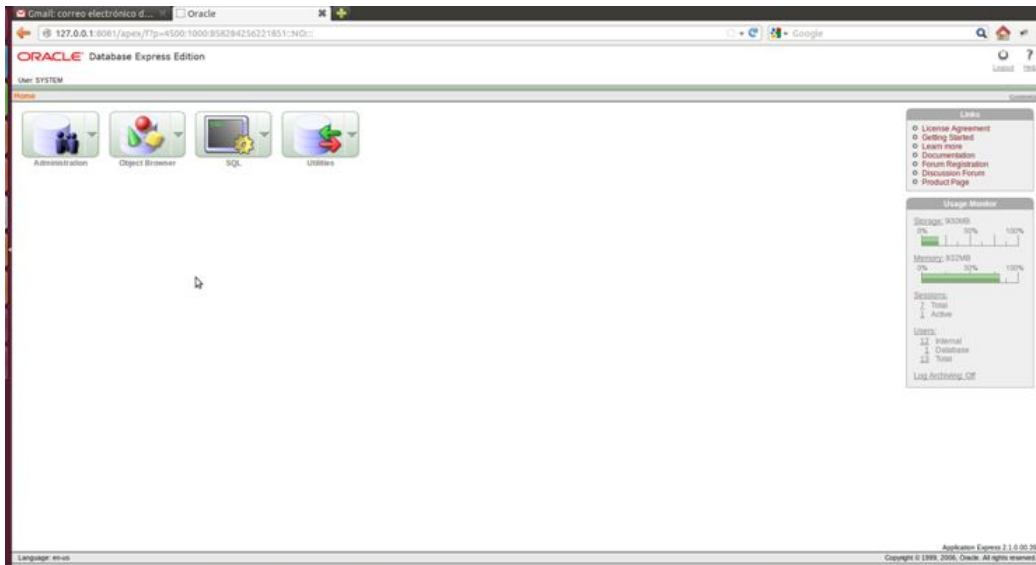
2.3.1 Base de datos

Dado que los datos estaban almacenados en una geodatabase (formato nativo de ArcGIS®), era necesario hacer un procesamiento para migrarlos a la base de datos Oracle. Los pasos que se siguieron fueron:

- Extraer *shapefiles*: utilizando el software SIG de escritorio ArcGIS®, se extrajeron los archivos que lo integran (.shp, .shx, .prj, .dbf) con la información correspondiente a las tablas de bloques, puntos de interés, espacio abierto, entre otros.
- Convertir los *shapefiles* en tablas Oracle, se utilizó la aplicación *Java Shapefile Converter* la cual es de uso gratuito se siguió el procedimiento explicado en (Oracle JSFConvert, 2009). Un ejemplo de instrucción es:

```
java -cp /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/ojdbc14.jar:/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/sdoutl.jar:/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/sdoapi.jar oracle.spatial.util.SampleShapefileToJGeomFeature -h 127.0.0.1 -p 1521 -s XE -u SYSTEM -d sysdba -f /home/usuario/shpCampus/campus/CMFC_PTS_INTERES -i gid -t cmfc_pint_tbcec -r 21892
```

A continuación se presentan algunas imágenes de la interfaz de administración de la base de datos, luego de realizados los dos pasos anteriores, donde se muestran las tablas generadas:



Object Browser
127.0.0.1:8081/apex/f?p=4500:1001:858284256221851::NO:RP:OB_CURRENT_TYPE:TABLE

ORACLE Database Express Edition

User: SYSTEM

Home > Object Browser

| Tables |
|--------------------------|
| AQS_INTERNET_AGENTS |
| AQS_INTERNET_AGENT_PRIVS |
| AQS_QUEUEUS |
| AQS_QUEUE_TABLES |
| AQS_SCHEDULES |
| CMFC_B18P01_TB |
| CMFC_B18P02_TB |
| CMFC_B18P03_TB |
| CMFC_B18P04_TB |
| CMFC_B18P05_TB |
| CMFC_B18P06_TB |
| CMFC_B18P07_TB |
| CMFC_BLOQUE3_TB |
| CMFC_ESPACIO_ABIERTO_TB |
| CMFC_PTS_INTERES_TB |
| CMT_DEPENDENCIA_ACAD |
| CUSTOMERS |
| DEFS_AQCALL |
| DEFS_AQERROR |
| DEFS_CALLDEST |
| DEFS_DEFAULTDEST |
| DEFS_DESTINATION |
| DEFS_ERROR |
| DEFS_LOB |
| DEFS_ORIGIN |
| DEFS_PROPAGATOR |
| DEFS_PUSHED_TRANSACTIONS |
| DEFS_TEMPLOB |
| ESTACION_METRO_TB |
| HELP |
| LINEA_METRO_TB |
| LOGMNRD_DBNAME_UID_MAP |
| LOGMNRD_GSII |
| LOGMNRD_GTCS |
| LOGMNRD_GTLO |
| LOGMNRD_CTAS_PART_MAP |
| LOGMNRD_MDN... |

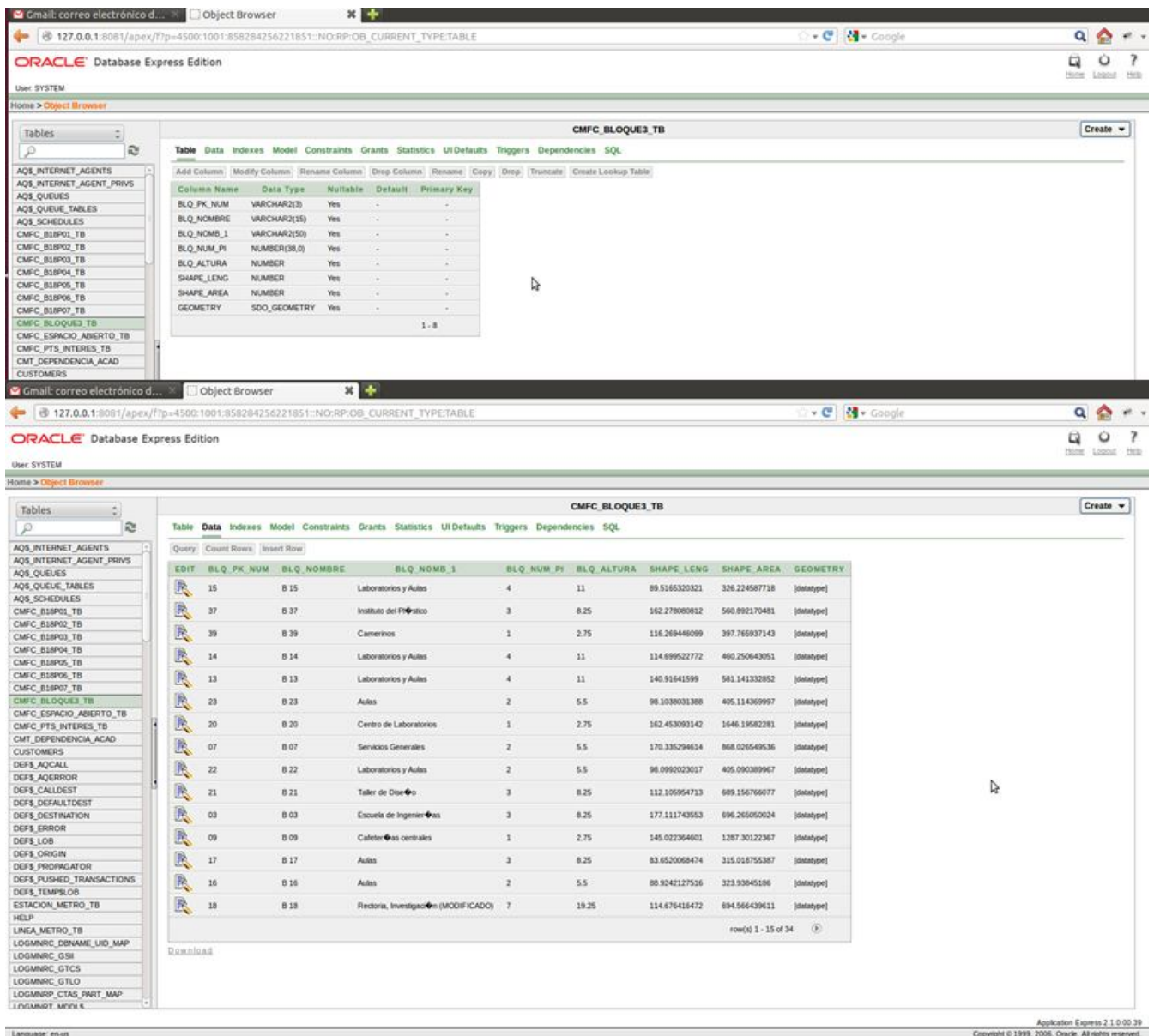


Figura 27. Base de datos Oracle con datos geográficos del campus EAFIT

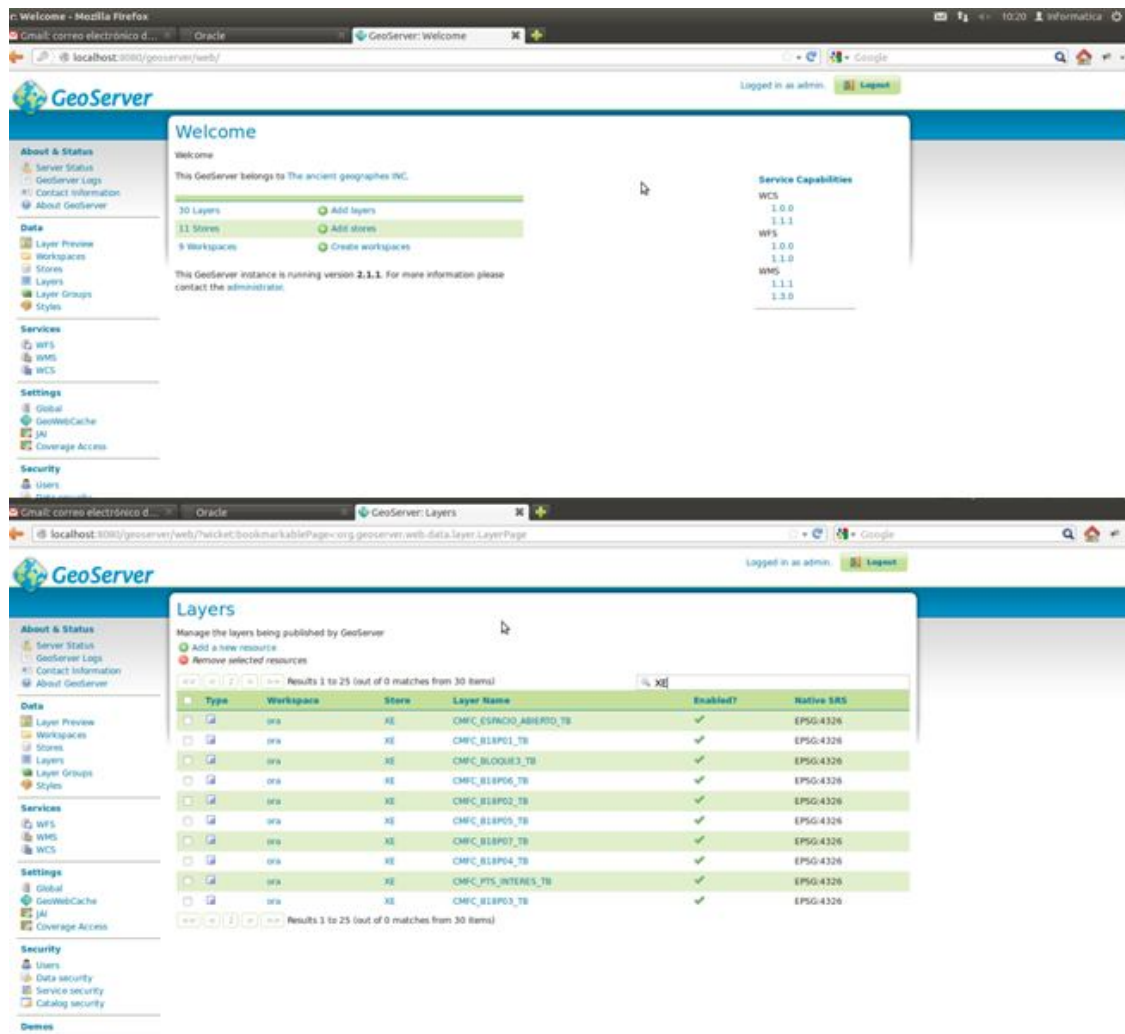
2.3.2 Publicación de los datos en Geoserver

Para publicar la información contenida en la base de datos se realizó el siguiente procedimiento, apoyado en la documentación de la página <http://docs.geoserver.org/latest/en/user/index.htm>

- Descargar e instalar *plugin* (componente adicional) en Geoserver, necesario para establecer la conexión con la base de datos Oracle.

- Crear *workspace* y configurar el *data-store* (parámetros de conexión tales como nombre de la base de datos, contraseña, puerto).
- Creación de capas: cada tabla se corresponde a un *layer* (capa). Se establecen sistemas de coordenadas y *bounding-boxes*. Después de esto las capas están listas para su consumo.
- Verificar que las capas estén correctamente publicadas mediante la utilidad de previsualización incluida en Geoserver.

El conjunto de imágenes a continuación da cuenta de esto:



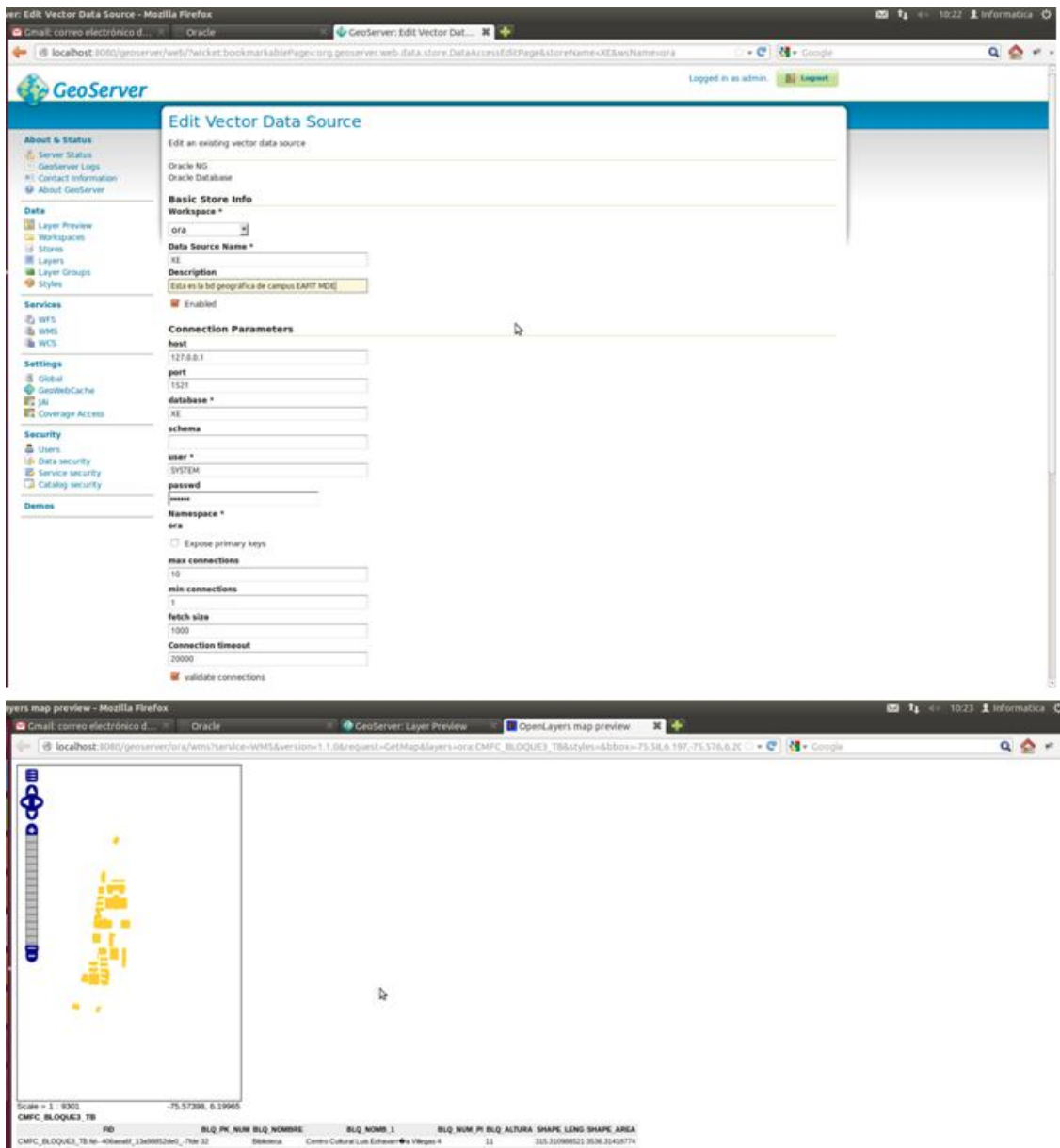


Figura 28. Geoserver con capas publicadas del campus EAFIT

2.3.3 Diseño e implementación de la aplicación

Con la base de datos y los datos publicados se inició el desarrollo de la aplicación piloto. La imagen a continuación muestra las terminales en Ubuntu sobre las cuales se observa el funcionamiento de Oracle, Geoserver y Apache Tomcat:

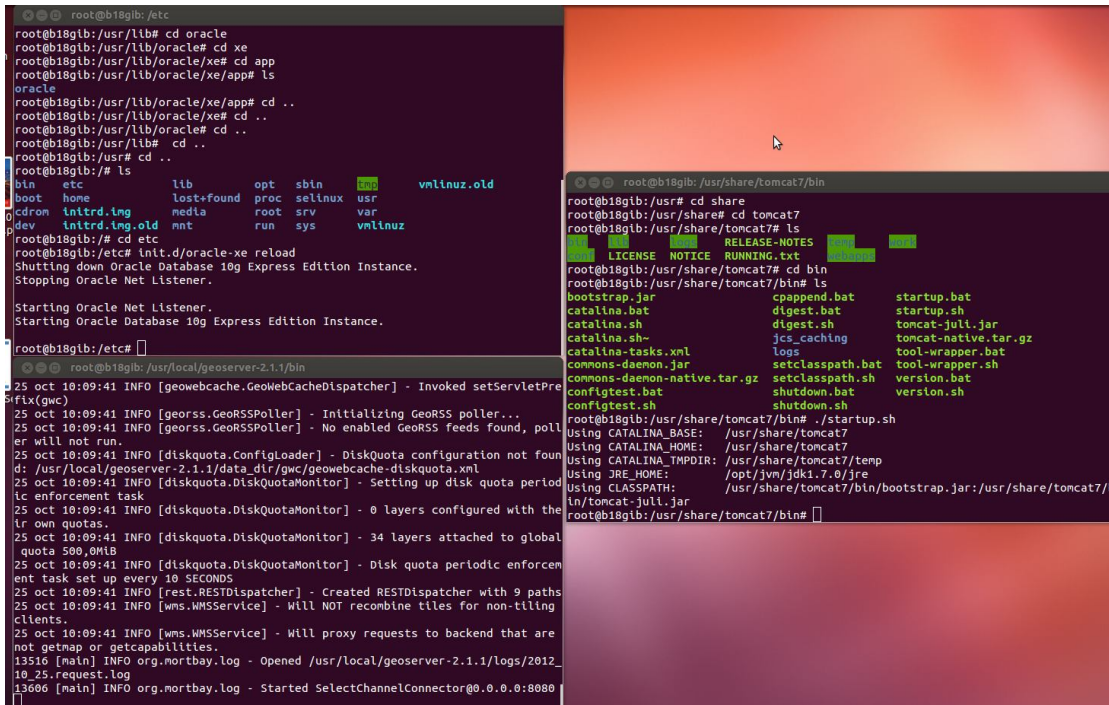


Figura 29. Terminales en Ubuntu con base de datos Oracle, Geoserver y Tomcat

De acuerdo con lo propuesto por Chandan (2011), la estructura de una aplicación que utiliza el *framework* ExtJS, es la que se muestra en la siguiente figura:



Figura 30. Estructura de una aplicación con ExtJS. Tomado de (Chandan, 2011)

Los principales componentes son:

- index.html: esta página contiene el código HTML encargado de traer las librerías de ExtJS, los CSS y los archivos JavaScript propios de la aplicación (JS).

- assets: contiene los recursos de imágenes y los archivos de estilo (css).
- lib: contiene las librerías de ExtJS y los JS de la aplicación.
- Main.js, se encuentra en pack1: es el punto de entrada a la aplicación, y de allí se llamará a los otros JS ubicados en los otros packs. Ejemplo:

```
Ext.onReady(function(){
    // Mandatory: need to add this spacer /*[relative path to..]*/

    Ext.BLANK_IMAGE_URL = 'lib/ext/resources/images/default/s.gif';
    Ext.QuickTips.init();

    // main execution start here, user defined function
    var onModuleLoad = function(){
        var viewLayout = new com.company.hello.package2.ViewLayout();
        viewLayout.createLayout();
    }

    // main starting point
    onModuleLoad();
});
```

Figura 31. Ejemplo de Main.js. Tomado de (Chandan, 2011)

- ViewLayout.js, ubicado en pack 2: es el encargado de estructurar la IU es decir, el *layout* de la aplicación. Un ejemplo de código es

```

Ext.ns("com.company.hello.package2");
(function(){
    var ViewLayout = Ext.extend(Object, {
        constructor: function(config) {
            Ext.apply(this, config);
        },

        // creating page layout, code goes here
        /* public */ createLayout: function(){
            // viewport panels
            var northPanel = new Ext.Panel({
                id: 'north-panel', height : 50, region: 'north', border: false, title: ' Top Panel'
            });

            var southPanel = new Ext.Panel({
                id: 'south-panel', height : 200, region: 'south', title : 'South Panel',
                collapsible: true, collapsed: true
            });

            var westPanel = new Ext.Panel({
                id: 'west-panel', layout: 'fit', width: 250, region: 'west', title: 'Navigation',
                collapsible: true
            });

            var centerPanel = new Ext.Panel({
                region: 'center' ,layout: 'fit', title: 'Content Panel'
            });

            // viewport
            new Ext.Viewport({
                id: 'id-viewport'
                ,layout : 'border'
                ,items : [northPanel, southPanel, westPanel, centerPanel]
            });
        }
    });

    com.company.hello.package2.ViewLayout = ViewLayout;
})();

```

Figura 32. Ejemplo de ViewLayout.js. Tomado de (Chandan, 2011)

Chandan, 2011 también sugiere una forma de crear clases en ExtJS, similar a los POJO (*Plain Old Java Object*), la siguiente figura muestra un ejemplo:

```

Ext.ns("com.company.app");
(function(){
    var JSClass = Ext.extend(Object,{
        //constructor function
        constructor: function(config){
            Ext.apply(this, config)
        }

        // class variable
        ,value1: null
        ,value2: null

        // getter, setter
        ,getValue1: function(){
            return value1;
        }
        ,setValue1: function(val){
            this.value1 = val;
        }
    })

    'com.company.app.JSClass = JSClass;
})();

```

Figura 33. Ejemplo Clase en ExtJS. Tomado de (Chandan, 2011)

La aplicación se construyó siguiendo esta misma organización, la imagen a continuación muestra la estructura:

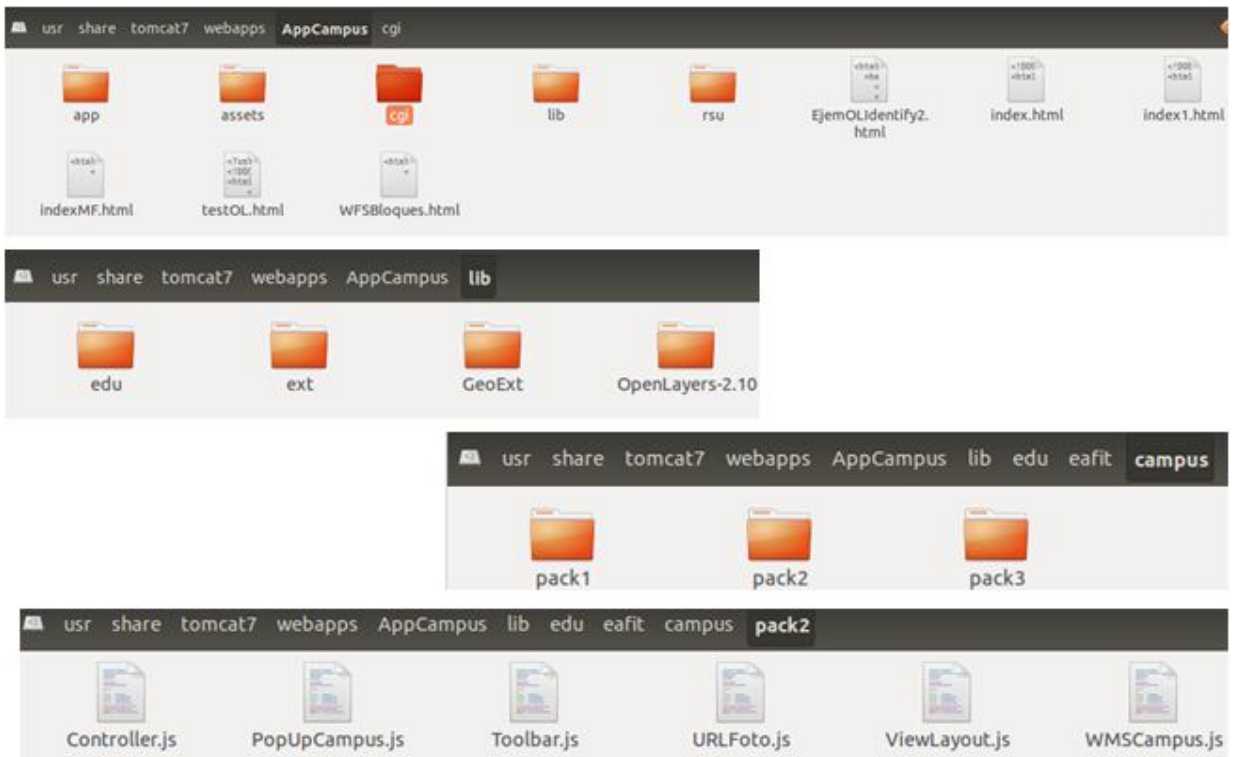


Figura 34. Organización de la aplicación con software libre

El esquema correspondiente es:

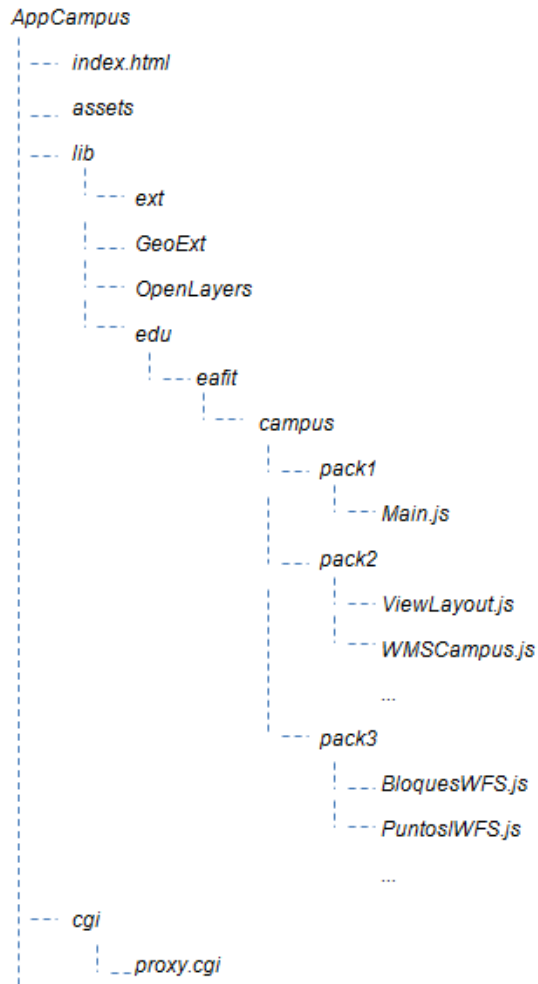


Figura 35. Esquema de la aplicación con software libre

En la estructura se puede observar el archivo `index.html`, la carpeta `lib`, que contiene el API `OpenLayers`, `GeoExt`, `Ext` y los JS de la aplicación, los cuales están conformados por `pack1`, `pack 2` y `pack3`.

`Pack2`: corresponde a las clases que se encargan de la vista (IU) y que controlan las interacciones con el usuario, éstas hacen uso de los *widgets* disponibles en `Ext` y `GeoExt`. Se encuentra `ViewLayout.js`, `WMSCampus` entre otras.

`Pack3`: se encuentran las clases que representan el modelo y permiten almacenar los datos que son traídos desde el servidor. Por ejemplo `BloquesWFS.js` y `PuntosIWFS.js`

La carpeta `CGI` contiene un script `proxy` que permite solicitar URLs a través de `AJAX` que no están en el mismo dominio. `OpenLayers` tiene disponible uno hecho en `python`, que se modifica según los sitios permitidos, vale anotar que este

mecanismo es adecuado para aplicaciones sencillas donde no se necesite mucha seguridad.

Las siguientes imágenes pertenecen a las diferentes funcionalidades implementadas:

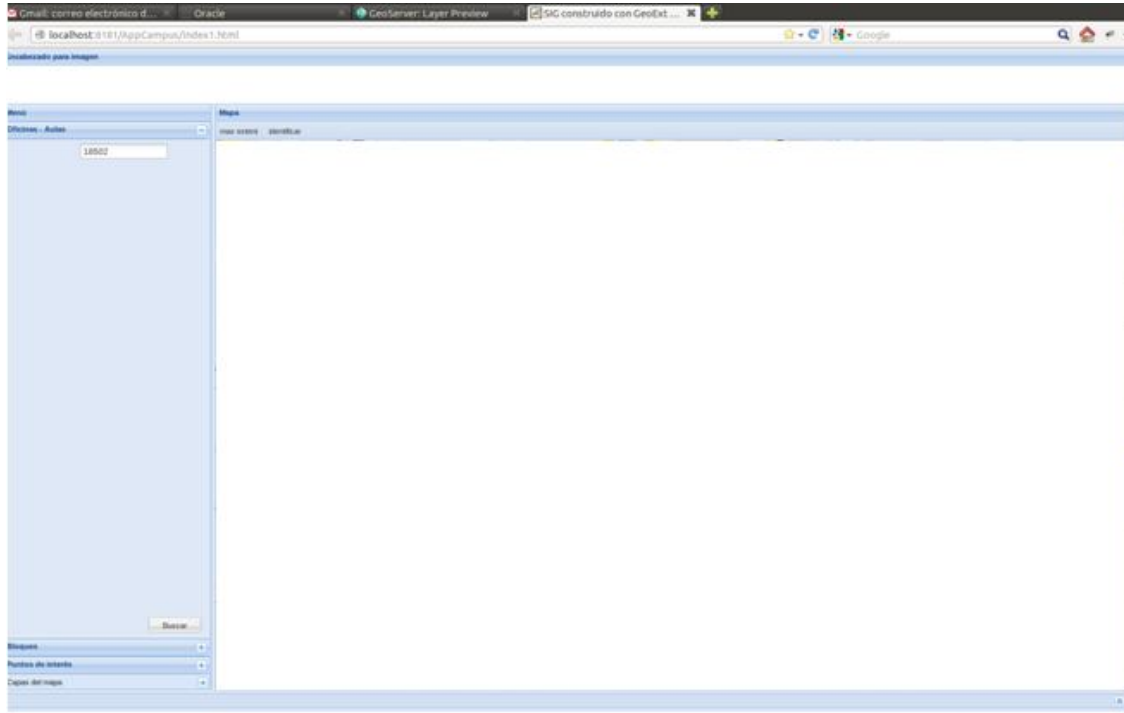


Figura 36. Layout generado con ExtJS y GeoExt

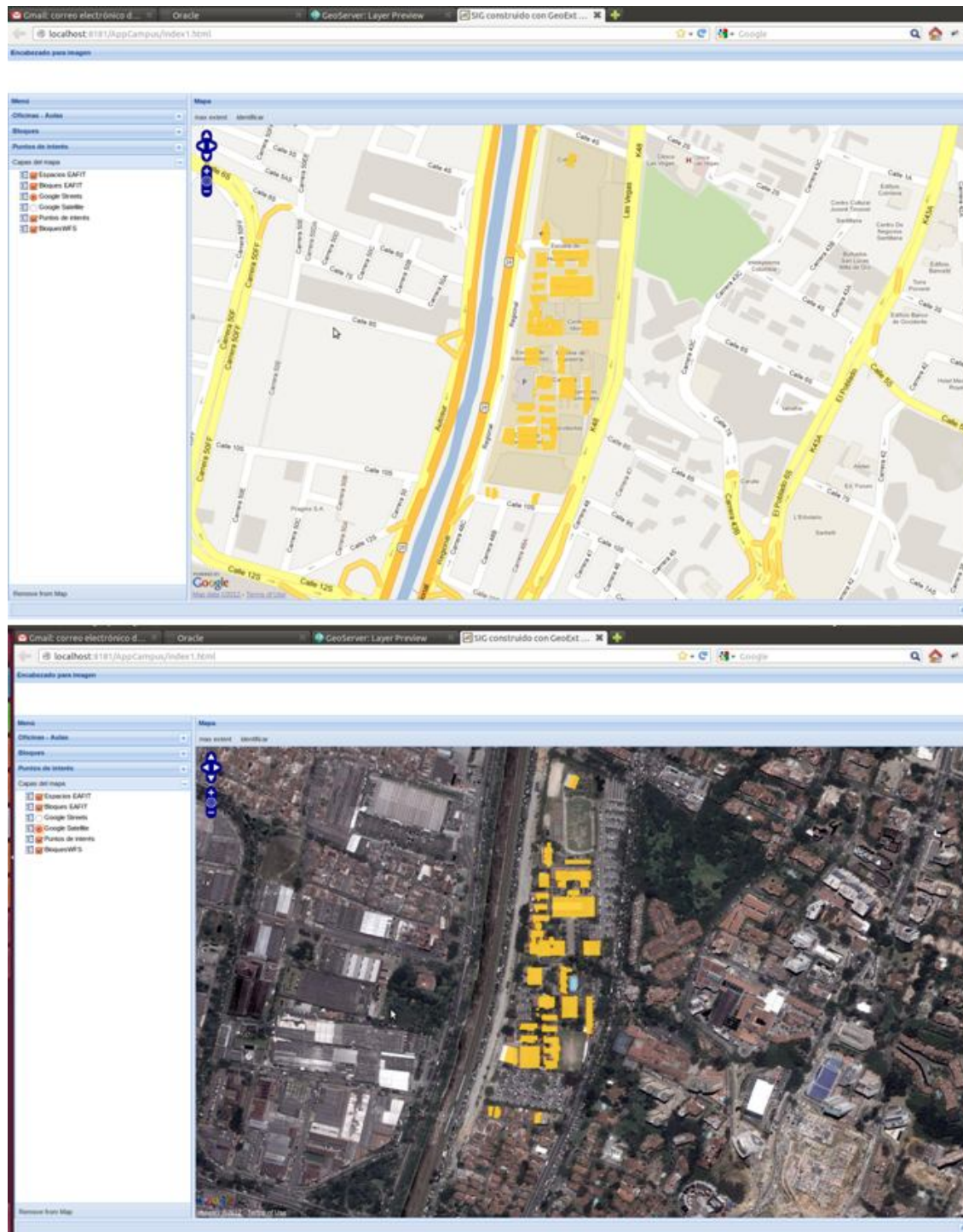


Figura 37. Página con mapa de Bloques, con fondo de Google® de calles y satélite

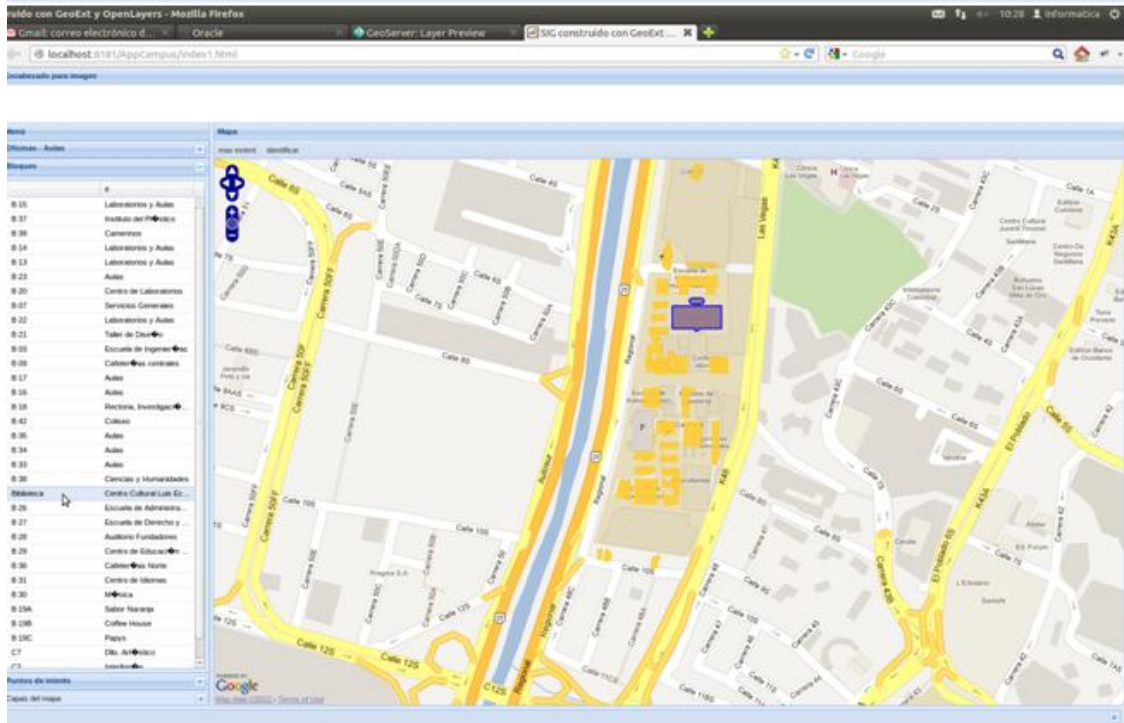
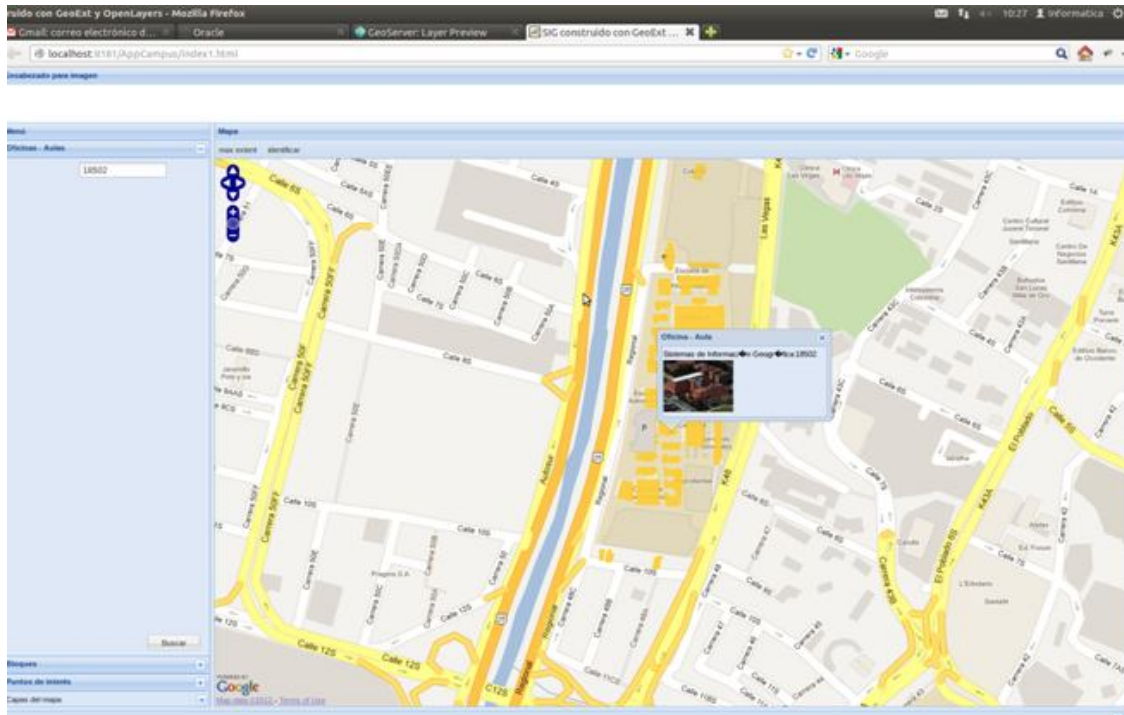


Figura 38. Funcionalidad de buscar oficina/aula (arriba) y localizar bloque (abajo)

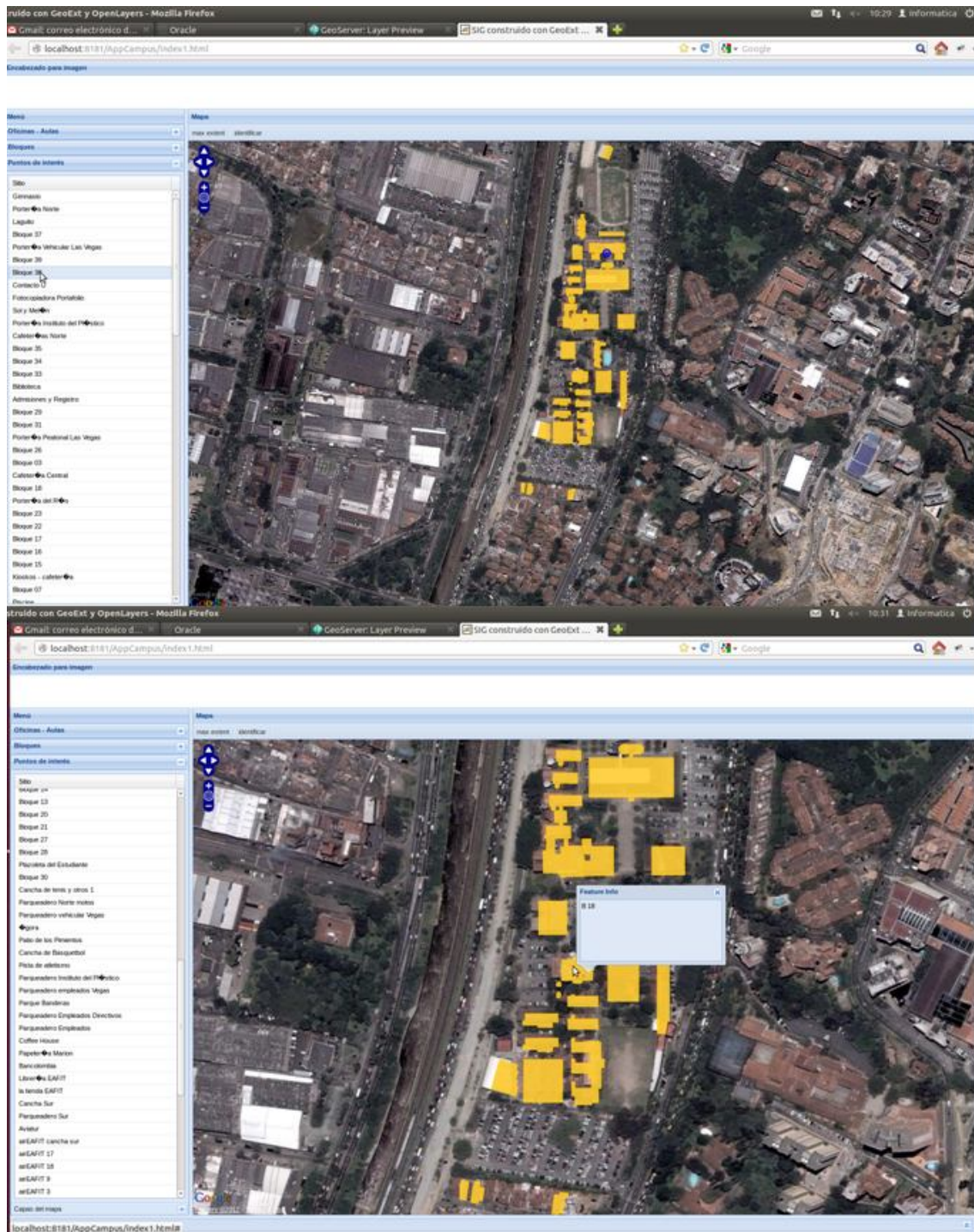


Figura 39. Funcionalidad de localizar puntos de interés (arriba) e identificar (abajo)

2.4 DESARROLLO DE APLICACIÓN PILOTO CON SOFTWARE PROPIETARIO

En la siguiente figura y tabla se ilustra la arquitectura y se resume el *stack* de herramientas seleccionadas:

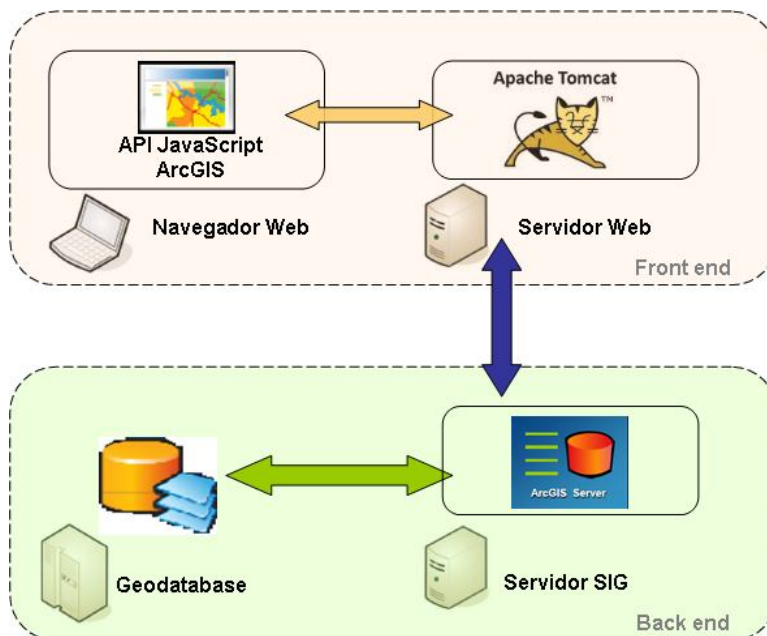


Figura 40. Arquitectura aplicación con software propietario

Tabla 30. Stack de herramientas para construir aplicación con software propietario

| | |
|-------------------|---|
| Sistema operativo | Windows 7 |
| Base de datos | Geodatabase (nativa) de Esri® |
| Servidor de mapas | ArcGIS Server v 9.3 |
| Servidor Web | Apache Tomcat 7 |
| Cliente | ArcGIS JavaScript API v.2.7. Basado en framework Dojo 1.6 |

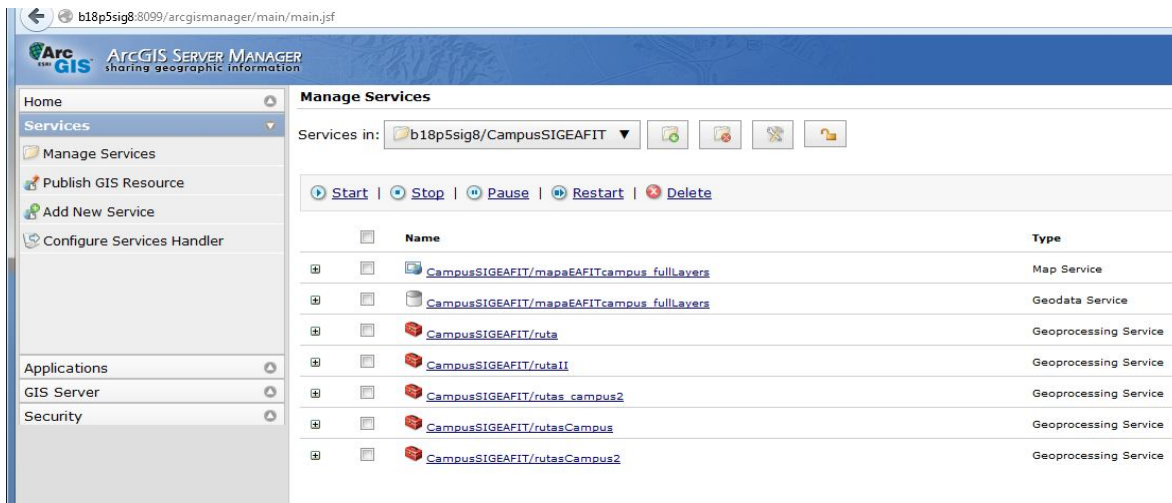
Siguiendo el orden de la arquitectura, primero se describe el proceso de construcción de la base de datos, luego la publicación en el servidor y finalmente la construcción de la aplicación:

2.4.1 Base de datos

Para utilizar Oracle se debe seguir un procedimiento semejante al realizado con el piloto de software libre y adicionalmente se necesita instalar un componente adicional denominado ArcSDE. Dadas estas condiciones y para aprovechar la facilidad que implica tener la base de datos y el servidor del mismo proveedor, se optó por trabajar con el formato nativo, es decir la Geodatabase.

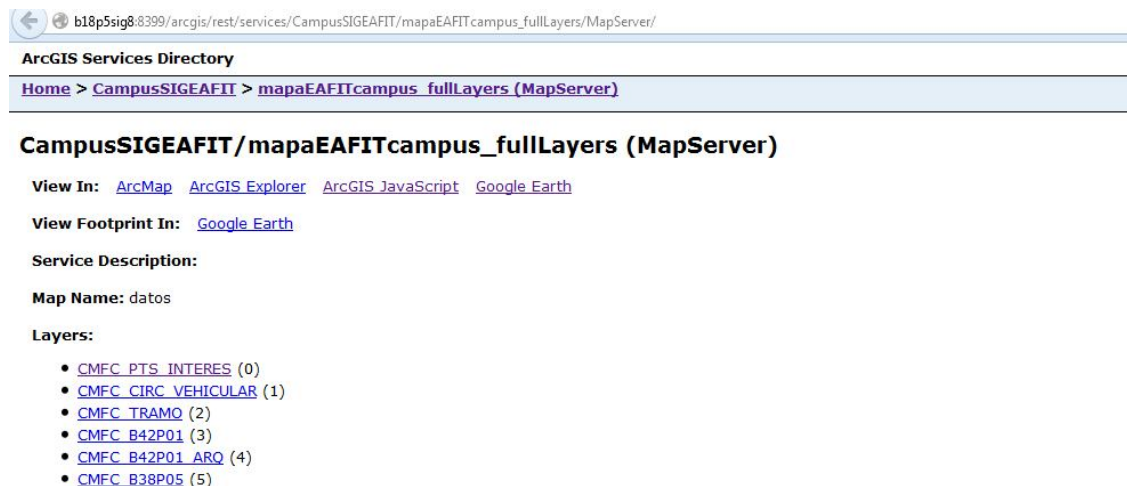
2.4.2 Publicación de los datos en ArcGIS Server

La publicación de la información contenida en la Geodatabase se realizó mediante la interfaz Web de ArcGIS Server, apoyado en la documentación de la página <http://webhelp.esri.com/arcgisserver/9.3/java/>. Las imágenes a continuación muestran las capas del campus EAFIT disponibles, además se señala que se pueden consumir vía los protocolos REST, SOAP, WMS y WCS.



The screenshot shows the ArcGIS Server Manager interface. The left sidebar contains navigation options: Home, Services (selected), Manage Services, Publish GIS Resource, Add New Service, and Configure Services Handler. The main area is titled "Manage Services" and shows the "Services in:" dropdown set to "b18p5sig8/CampusSIGEAFIT". Below this are control buttons: Start, Stop, Pause, Restart, and Delete. A table lists the following services:

| Name | Type |
|---|-----------------------|
| CampusSIGEAFIT/mapaEAFITcampus_fullLayers | Map Service |
| CampusSIGEAFIT/mapaEAFITcampus_fullLayers | Geodata Service |
| CampusSIGEAFIT/rutas | Geoprocessing Service |
| CampusSIGEAFIT/ruta1 | Geoprocessing Service |
| CampusSIGEAFIT/rutas_campus2 | Geoprocessing Service |
| CampusSIGEAFIT/rutasCampus | Geoprocessing Service |
| CampusSIGEAFIT/rutasCampus2 | Geoprocessing Service |



The screenshot shows the ArcGIS Services Directory page for the service "CampusSIGEAFIT/mapaEAFITcampus_fullLayers (MapServer)". The page includes the following information:

- View In:** [ArcMap](#), [ArcGIS Explorer](#), [ArcGIS JavaScript](#), [Google Earth](#)
- View Footprint In:** [Google Earth](#)
- Service Description:**
- Map Name:** datos
- Layers:**
 - [CMFC PTS_INTERES](#) (0)
 - [CMFC_CIRC_VEHICULAR](#) (1)
 - [CMFC_TRAMO](#) (2)
 - [CMFC_B42P01](#) (3)
 - [CMFC_B42P01_ARQ](#) (4)
 - [CMFC_B38P05](#) (5)

- [CMFC_B03P02](#) (179)
- [CMFC_B03P02_ARQ](#) (180)
- [CMFC_B03P01](#) (181)
- [CMFC_B03P01_ARQ](#) (182)
- [CMFC_ESPACIO_ABIERTO](#) (183)
- [CMFC_BLOQUE](#) (184)
- [CMFC_CAMPUS_MED](#) (185)
- [EntornoGrafico](#) (186)
- [IOB_subescena_Campus.png](#) (187)

Description:

Copyright Text:

Spatial Reference: 21897

Single Fused Map Cache: false

Initial Extent:

XMin: 833655.667125
 YMin: 1177228.772275
 XMax: 834303.716175
 YMax: 1178058.694225
 Spatial Reference: 21897

Full Extent:

XMin: 833376.638039225
 YMin: 1176969.82767513
 XMax: 834571.149183568
 YMax: 1178716.76726261
 Spatial Reference: 21897

Units: esriMeters

Document Info:

- Title: **RUTACampus_27sep2007**
- Author: **SUSANA_**
- Comments:
- Subject:
- Category:
- Keywords:

Supported Interfaces: [REST](#) [SOAP](#) [WMS](#) [WCS](#)

Supported Operations: [Export Map](#) [Identify](#) [Find](#) [Generate KML](#)

Figura 41. Interfaz de administración de ArcGIS Server 9.3 con datos publicados del Campus EAFIT.

2.4.3 Diseño e implementación de la aplicación

Dojo es el framework sobre el que se soporta el API ArcGIS JavaScript, entonces, primero se hace necesario conocer que está constituido por tres componentes:

- *Dojo* o núcleo: encargado de la normalización del navegador, carga de módulos, manipulación del DOM, eventos, funciones de *string*, acceso a datos, llamadas XHR, codificación/decodificación de JSON.
- *Dijit*: son los widgets Dojo. Un widget encapsula la representación del DOM y ofrece una interfaz orientada a objetos. Diferentes tipos: de layout, forms/validación, menús
- *Dojox*: conformado por varios proyectos extra, en parte experimentales. Incluye otros *widgets*, efectos de animación, validación de esquemas JSON, entre otras funcionalidades.

En el sitio de [sitepen](#) se da un ejemplo de clases en Dojo, la cual tiene una estructura del tipo:

```
dojo.provide('davidwalsh.Menu');

dojo.declare('davidwalsh.Menu',dijit.Menu,{
```

```

/* propiedades y métodos */
myCustomProperty: true,
myCustomMethod: function() {
    /* funcionalidad... */
}
// todos los métodos de dijit.Menu también forman parte de esta clase, se heredan
});

```

En la creación de una clase se emplean las sentencias:

- `dojo.require`: indica la dependencia de un módulo particular.
- `dojo.provide`: se utiliza para que Dojo reconozca un archivo .js como un módulo hace que los recursos (clases, funciones o colección de `dojo.require`) contenidos en un archivo estén disponibles
- `dojo.declare`, es la base para creación de clases, permite herencia múltiple.

La estructura de la aplicación sigue el esquema:



Su contenido es:

- `index.html`: página que contiene los llamados al API desde el sitio que ESRI tiene dispuesto para esto (<http://serverapi.arcgisonline.com>). Y se establece el *layout* mediante los widgets de Dijit (i.e. `BorderContainer`, `ContentPane`).
- `css`: contiene los archivos de estilo.
- `images`: contiene las imágenes utilizadas en la aplicación.
- `js`: contiene los archivos JavaScript propios de la aplicación. Dentro los JS se encuentran:

base.js: es una clase controladora, se construyó basada en lo propuesto por Foster, (2011). Esta se encarga de llamar a las otras clases encargadas de funcionalidades como identificar, mapa (que trae la capa de base y la de bloques), entre otras

Las siguientes imágenes ilustran la estructura:

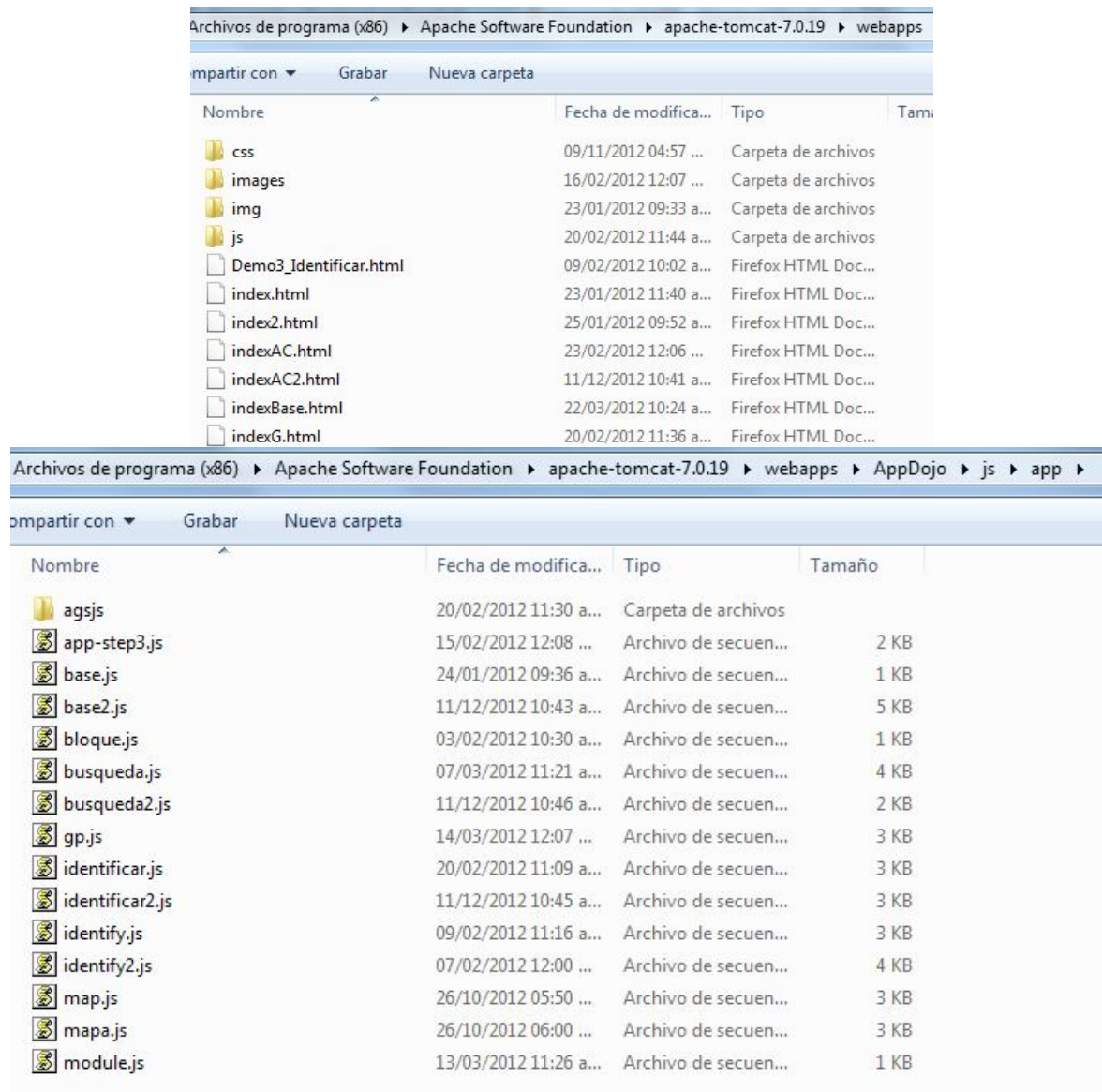


Figura 42. Organización de la aplicación con software propietario

Las imágenes a continuación pertenecen a diferentes funcionalidades implementadas en el piloto:

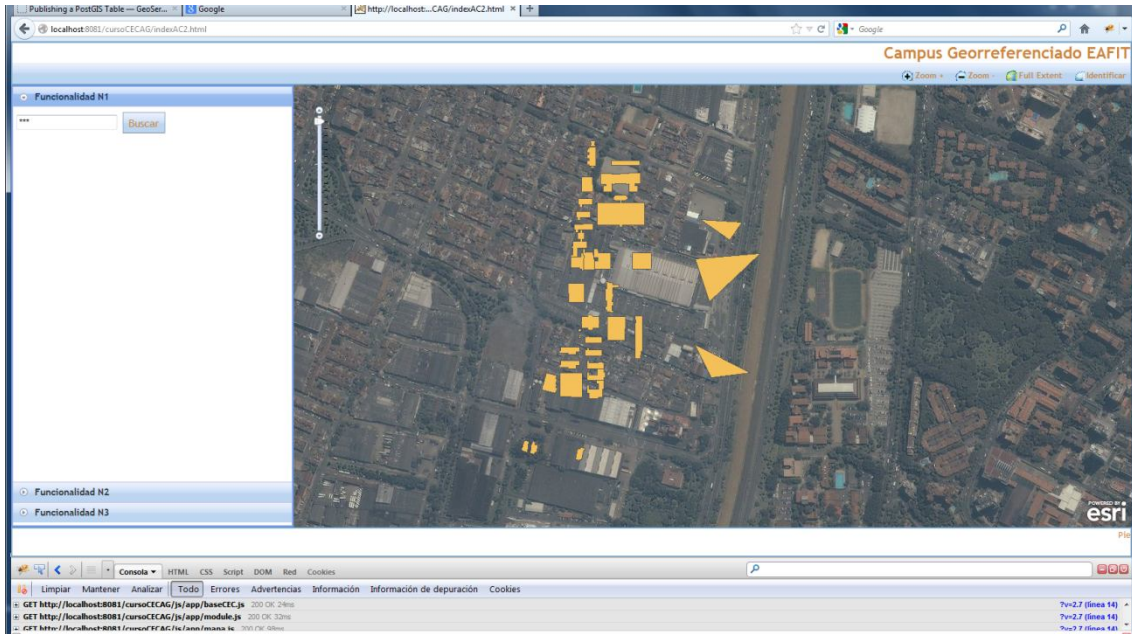


Figura 43. Layout de la página y mapa con imágenes de fondo desde servidor ESRI®

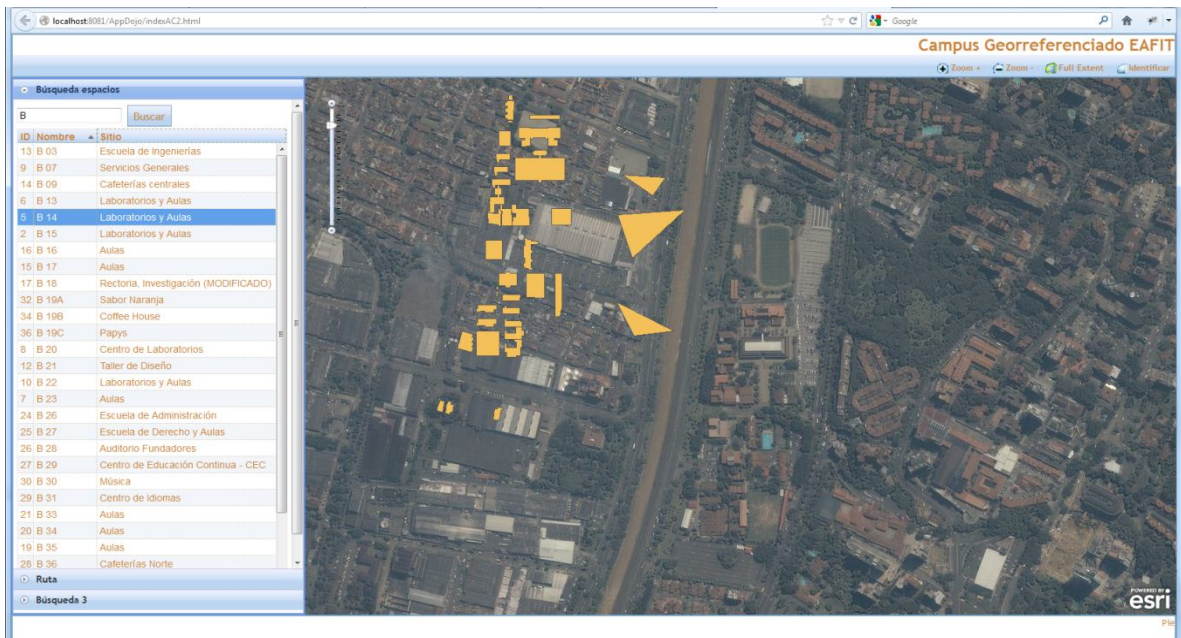


Figura 44. Funcionalidad de búsqueda por bloques

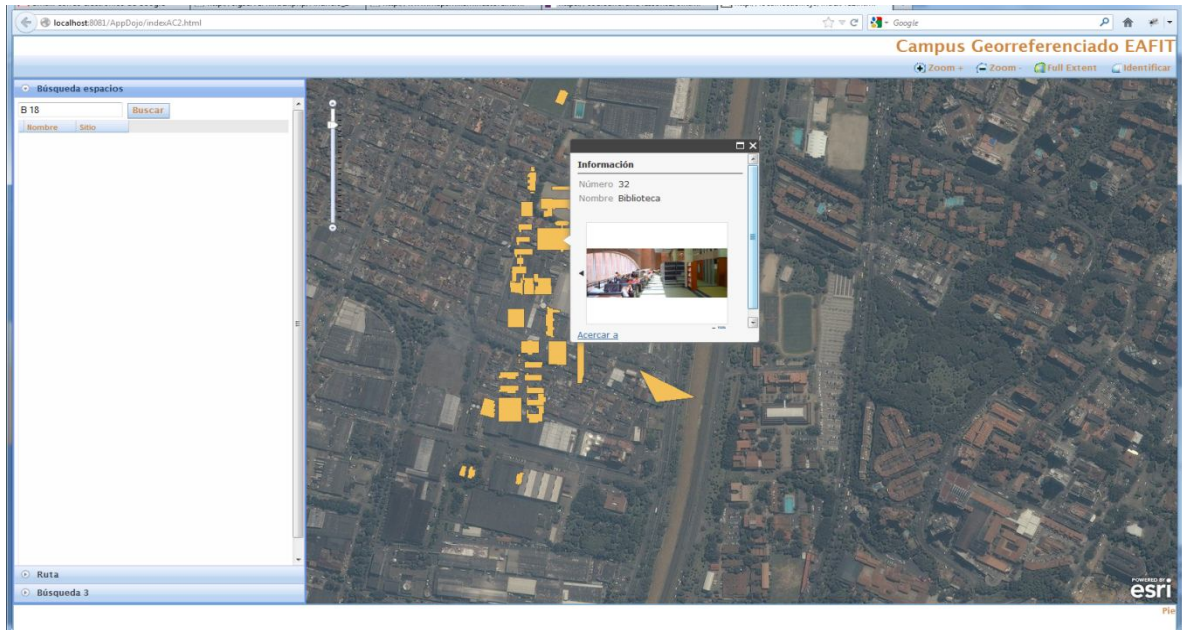


Figura 45. Funcionalidad de identificar

3 CAPITULO 3

LINEAMIENTOS PARA EL DESARROLLO DE APLICACIONES SIG WEB

Lo descrito en este capítulo es el resultado del estudio teórico y de la experiencia obtenida en desarrollos anteriores y durante la construcción de los pilotos. Los lineamientos se dividieron en tres etapas: antes, durante y después del proyecto.

3.1 ANTES DEL DESARROLLO DEL PROYECTO

- Dimensionar la magnitud del desarrollo, en términos de funcionalidad: hay diferencia entre una aplicación sencilla en la que sólo se desea embeber un mapa en una página Web -con funcionalidad básica como tener controles de zoom, paneo o listar capas- a crear una en la que sean múltiples funcionalidades de mediana - alta complejidad (i.e. trazado de rutas, generación de nueva información geográfica, consultas avanzadas). En el primer caso, por ejemplo, el uso de la librería OpenLayers es suficiente; en el segundo, el trabajo debe ser más cuidadoso, se debe tener una arquitectura, emplear patrones y frameworks, que faciliten el desarrollo, trabajo en equipo y reuso de código. Sin embargo, por simple que parezca la aplicación, se puede aplicar la estructura más básica que consiste en separar HTML, estilos y código, esto da pie a que continúe su crecimiento. Otro aspecto importante es la modularización del código, que puede realizarse a varios niveles, la siguiente tabla muestra un ejemplo, desde la elemental, hasta una con mayor elaboración:

Tabla 31. Niveles de modularización de código

| Elemental | Media | Avanzada |
|---|---|--|
| <pre><script type="javascript" src="jsConGlobales1.js" ></script></pre> | <pre>//aplicar Module Pattern <script type="javascript" src="Modulo1.js"></scrip t></pre> | <pre>// aplicar AMD en Dojo define(// lista de dependencias</pre> |
| <pre><script type="javascript" src="jsConGlobale2.js"> </script></pre> | <pre><script type="javascript" src="Modulo2.js"></scrip t></pre> | <pre>['pkgA/modA', 'pkgA/modB', 'pkgZ/modC'],</pre> |
| <pre><script type="javascript" src="jsConGlobalesN.js" ></script></pre> | <pre><script type="javascript" src="Modulo3.js"></scrip t></pre> | <pre>// definición funciones del módulo</pre> |
| <pre>// resultado</pre> | <pre>//...</pre> | <pre>// dependencias se mapean a parámetros de fx</pre> |
| <pre><script type="javascript" src="muchoTiempoCarg a.js"></script></pre> | <pre><script type="javascript" src="ModuloN.js"></scri pt></pre> | <pre>function (modA, modB, modC) {</pre> |

| | | |
|--|---|---|
| | <pre>// resultado <script type="javascript" src="aunMuchoTiempoC arga.js"></script></pre> | <pre>// se crea el módulo var myModule = { doStuff: function() { ... } }; // se retorna el módulo return myModule; });</pre> |
|--|---|---|

- Tomar conciencia que el desarrollo de aplicaciones del lado del cliente tiene el mismo rigor que las tradicionales del lado del servidor (en Java o .Net), en especial cuando se utiliza JavaScript. La siguiente imagen es un llamado de atención, a evitar el crecimiento desordenado o código spaghetti:



Figura 46. Código spaghetti. Tomado de: <http://www.enterprisedojo.com/2010/08/24/writing-modular-javascript-with-doj/>

- Preparar la conformación del equipo de trabajo, el cual debe contar con varios perfiles de dos áreas de conocimiento, de parte de la rama de los SIG y de la ingeniería.

Los roles que se pueden establecer son los siguientes:

- Director del proyecto: debe conocer de SIG y de desarrollo de software. Hace el seguimiento, toma decisiones concernientes al proyecto y es responsable de la terminación del producto a satisfacción del cliente.
- Profesionales SIG: encargados del levantamiento de datos en campo, manejo de SIG de escritorio, elaboración de cartografía y productos que cumplan con las normas de calidad, estándares y métrica.

- Arquitecto: igual que el director, debe tener conocimientos en las dos áreas, enfatizando en la parte software, es el encargado de definir la estructura general del sistema y su modularización, debe estar al tanto de diversos *frameworks* y de patrones que se emplean a nivel genérico y los de uso particular.
 - Diseñador de interfaz y de experiencia de usuario: su papel es muy importante, ya que la parte visual y la interacción del usuario es determinante en este tipo de aplicaciones. Debe tener buena fundamentación en cuanto a aspectos de usabilidad.
 - Desarrolladores: deben contar con nociones de SIG y Geoservicios, elaboran código fuente basado en la arquitectura planteada, aplican patrones y buenas prácticas en la implementación, hacen pruebas unitarias de la funcionalidad que tienen a su cargo. También pueden plantear sugerencias para mejorar las características de la aplicación.
 - Integrador(es): encargado de ensamblar las partes funcionales que entregan los desarrolladores.
 - Probador(es): planea, diseña y evalúa las pruebas para asegurar el funcionamiento y la calidad del producto. Verifica que la entrega integrada funciona correctamente en un entorno parecido al de ejecución del sistema, determina cuándo el producto está listo para el despliegue
- Como en todo proyecto, evaluar si el equipo de trabajo tiene los conocimientos necesarios, en especial si es la primera vez que se enfrenta un desarrollo de este tipo. Es importante para definir si se establece un plan de capacitaciones y en qué áreas enfocarse, por ejemplo en patrones, frameworks, manejo de datos geográficos/geodesia o metodologías. Es deseable que los desarrolladores tengan experiencia con HTML, CSS, JavaScript básico y les sea familiar la programación orientada a objetos. Los cursos cortos, ya sean presenciales o virtuales son una opción para instruir al personal e incentivarlo para que utilicen buenas prácticas y eviten errores comunes, los recursos que demanda esta tarea se ven compensados con un proceso de desarrollo más limpio y quienes están acostumbrados a programar asimilan con facilidad este tipo de temáticas.
 - Revisar las metodologías que se ajusten a las características del proyecto (tiempo, presupuesto, personal). Una buena alternativa son las ágiles, ya que se adaptan a este tipo de aplicaciones que requieren que se muestren resultados rápidamente.
 - Hacer un estudio o barrido sobre las tecnologías vigentes, esto es de especial importancia ya que ayuda a mitigar riesgos en el desarrollo y a mejorar las perspectivas de extensibilidad y mantenibilidad. Las consultas pueden hacerse en blogs, publicaciones especializadas y en las propias páginas de las herramientas, también es de utilidad asistir a jornadas, encuentros o

presentaciones académicas y comerciales de las herramientas. Factores que pueden tenerse en cuenta son:

- Fecha del último *release*: indica si está actualizado o está en vía de ser abandonado.
- Tiempo entre cada *release*: muestra el grado de trabajo y evolución que hay en el proyecto
- Fecha prevista de lanzamiento de la próxima versión: si es pronto, quizá es prudente esperar a las últimas mejoras que utilizar la corriente.
- Tipo de licenciamiento: si es de código abierto o propietario y condiciones especiales.
- Dependencia de otros *frameworks*, incluyendo su influencia directa sobre el tipo de licencia, el panorama de cambio, ya que algunos pueden presentar saltos abruptos en el manejo. Es importante analizar si los frameworks proponen una aplicación de referencia (o mecanismo similar a los *blueprints* de Java).
- Lenguaje que utiliza: apunta al grado de madurez de la tecnología y recepción por parte de la comunidad desarrolladora.
- Propietario u organización que lidera el desarrollo: da idea del soporte que brinda.
- Actividad que tiene en los foros o listas de correo: esto señala el grado de acogida entre la comunidad desarrolladora y si es fácil conseguir ayuda sobre la herramienta.
- Participación en encuentros de orden mundial, regional o local en los que se promoció su uso: pone de manifiesto la aceptación y uso que está teniendo en la comunidad, además del nivel de difusión de la herramienta.
- Cantidad y características de proyectos que se hayan desarrollado con las herramientas (i.e. gubernamentales, de propósito específico o único, complejidad de las funcionalidades implementadas).

La siguiente tabla da un ejemplo del lineamiento anterior para los casos concretos de GeoExt y el API JavaScript de ArcGIS:

Tabla 32. Ejemplo de lineamiento, para GeoExt y API JavaScript de ArcGIS

| | GeoExt | API JavaScript ArcGIS |
|---|--|--|
| Tipo de licenciamiento | <p>ExtJS, framework del cual depende directamente, es licenciado bajo GPL, siempre y cuando la aplicación en la que se emplee preserve los términos aquí pactados.</p> <p>Si es una aplicación comercial y no se puede liberar el código, se debe adquirir una licencia (por cada uno de los desarrolladores del equipo) a la compañía Sencha (propietario de ExtJS). El pago no incluye soporte y es único.</p> | <p>El API puede utilizarse gratuitamente desde el sitio Web donde se encuentra alojado (<i>hosted version</i>) y también puede ser descargado para su uso local. Sin embargo la potencialidad de esta tecnología se explota en su totalidad cuando se emplea en conjunto con un servidor ArcGIS, ya que implementa una diversidad de funcionalidades bajo el protocolo REST (comunicación nativa).</p> |
| Dependencia de APIs, frameworks y plataformas | <p>GeoExt 1.1 utiliza la versión 3.4 de Ext JS.</p> <p>Sencha liberó la versión 4, la cual orienta la creación de las aplicaciones mediante un patrón arquitectural MVC (modelo vista controlador), prometiendo un proceso de desarrollo más eficiente.</p> <p>Desafortunadamente las versiones 3.4 y 4 no son compatibles, por lo que sus ventajas no pueden ser aprovechadas.</p> <p>Pero hay un proceso de migración de GeoExt, hacia uno nuevo, llamado GeoExt 2</p> | <p>El API ha evolucionado, por lo cual ha cambiado de versiones del framework Dojo, que van desde la 1.1.0 hasta la 1.7.3</p> <p>En el caso de las versiones 2.3 a la 2.8 del API, se emplea Dojo 1.6, el cual utiliza un cargador síncrono y el sistema de empaquetamiento, se caracteriza por las siguientes cláusulas: <i>dojo.require</i>, <i>dojo.provide</i> y <i>dojo.declare</i>.</p> <p>La mayoría de ejemplos disponibles del API emplean la notación anterior.</p> <p>A partir de la versión 3.0 del API,</p> |

| | | |
|--|--|---|
| | <p>(con ExtJS 4), que aunque requiere de mucho trabajo, el equipo promotor de esta herramienta, ya ha realizado adelantos mediante <i>code sprints</i> logrando liberar una versión alfa en mayo de 2012.</p> <p>Para versiones anteriores a la 4 de ExtJS se encuentran arquitecturas y estructuras planteadas por terceros, ya sean expertos o consultores, tal es el caso de la propuesta hecha por Chandan (2011). Sencha, en sus ejemplos muestra el uso aislado de los componentes, no se expone su uso en conjunto.</p> | <p>se utiliza Dojo 1.7, que tiene una filosofía diferente de carga y de estructuración, llamada Asynchronous Module Definition (AMD). Aunque esta versión soporta la anterior forma de trabajo (<i>legacy</i>), cuando Dojo pase a 2.0, no tendrá esta capacidad.</p> <p>Los ejemplos de utilización de AMD con el API son escasos.</p> <p>Por otro lado, se pueden presentar incompatibilidades entre la parte servidora y el API. Por ejemplo los casos de ArcGIS Server 10.1 (Beta1, Beta 2, Pre-release) no soportan la versión 3.0 del API.</p> <p>Para la versión de Dojo que emplea AMD, se dispone de una plantilla que puede utilizarse para generar la arquitectura de la aplicación (https://github.com/csnover/dojo-boilerplate). En las anteriores versiones, no se cuenta con este mecanismo, Dojo simplemente provee los componentes, pero no hay un mecanismo que permita ensamblarlas.</p> <p>Son pocos los ejemplos que se encuentran sobre implementación de patrones en Dojo, la página oficial dispone de algunos puntuales. Se encuentra información útil en sitios de terceros, como por ejemplo en el de la compañía sitepen. En consecuencia, depende del arquitecto software y de los desarrolladores seleccionar e idear la forma de implementarlos.</p> |
|--|--|---|

| | | |
|---|---|--|
| <p>Participación en eventos</p> | <p>Presentan workshops en el FOSS4G.</p> <p>En las Jornadas de SIG libre organizadas por el SIGTE, de la Universidad de Girona (http://www.sigte.udg.edu/jornadassiglibre/) se exponen proyectos en los que se ha utilizado.</p> | <p>Conferencias de usuarios que organiza ESRI a nivel internacional y local. Donde se demuestran algunas capacidades de las tecnologías, áreas donde se aplica y se puede preguntar al personal técnico de la compañía.</p> |
| <p>Aplicaciones desarrolladas bajo la herramienta</p> | <p>TriMet: Public Transit in the Portland Area. http://trimet.org</p> <p>Vissir 3 http://www.icc.cat/vissir/index.html?lang=esl</p> | <p>Savannah Live well & prosper http://availableproperty.seda.org/</p> <p>St. Louis County Crime Incident Map http://maps.stlouisco.com/police/index.html</p> |

- Unido al anterior lineamiento, si se tienen los recursos disponibles, es oportuno realizar pruebas de concepto o pilotos, o utilizar demos de productos, que permitan dar un panorama global del manejo de las posibles tecnologías a emplear y los conceptos que manejan. Por ejemplo, la firma OpenGeo tiene a disposición una suite, (*trial* de 30 días, conformada por PostGIS, Geoserver, GeoWebCaché, OpenLayers y GeoExt); ArcGIS también tiene visores que se pueden descargar.
- Una excelente aproximación a las tecnologías y a los conceptos a tener en cuenta, es probar aplicaciones desarrolladas bajo las herramientas candidatas a utilizar, es decir, actuar como usuario, de esta manera se aprecia la parte gráfica o visual, la velocidad de respuesta, usabilidad, tipo de funcionalidades implementadas, entre otras particularidades.
- Es recomendable que parte del equipo o todo, asista a eventos de orden académico y comercial, relacionados con esta temática, de esta manera puede interactuar con desarrolladores, proveedores de tecnología y usuarios. Estos espacios son valiosos ya que puede presentar inquietudes y conocer de primera mano las experiencias de otros proyectos.
- Si se tiene un cliente potencia, se debe indagar cuáles son las políticas sobre el uso de software de código abierto y propietario, ya que esto restringe las opciones en cuanto a tecnologías.

A continuación se encuentran una serie de parámetros básicos para el desarrollo de una aplicación SIG Web, del lado del cliente (*front-end*), según tres criterios:

- a) Herramienta a utilizar.
- b) Estructura.
- c) Prácticas de metodologías ágiles que se pueden emplear.

Calificados según el tipo de funcionalidad, sencilla o avanzada (grado de complejidad) a implementar.

Tabla 33. Parámetros básicos DASW

| | | Tipo aplicación – características | |
|---------------------------------------|--|---|--|
| | | Sencilla: | Avanzada: |
| Parámetro | | <p>Mapa embebido en una página Web, es un complemento visual al resto de información.</p> <p>Funcionalidad básica como: zoom in, zoom out, paneo, encender apagar pocas capas</p> | <p>Mapa es el punto focal de la página.</p> <p>Múltiples funcionalidades y son de mediana - alta complejidad, como: consultas avanzadas (topónimos o atributos), trazado de rutas, análisis espaciales</p> |
| Herramientas (librerías y frameworks) | OpenLayers | ✓ | |
| | API JavaScript ArcGIS Server® (Basado en Dojo) | ✓ | ✓ |
| | OpenLayers + GeoExt (Basado en ExtJS) | | ✓ |

| | | | |
|--|---|---|---|
| Estructura del código | Separación básica: HTML + estilo + código [1] | ✓ | ✓ |
| | Encapsulación (clases orientadas a objetos) [2] | ✓ | ✓ |
| | Aplicación de patrón arquitectónico [3] | | ✓ |
| Prácticas de metodología ágiles | Requisitos con historias de usuario | | ✓ |
| | Requisitos consignados en <i>backlog</i> | | ✓ |
| | Diseño de arquitectura (documento general) | | ✓ |
| | Integración continua | | ✓ |
| | Aplicación de test funcionales con el usuario | ✓ | ✓ |

[1] Un ejemplo de separación o modularización básica se observa en la siguiente figura:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"></meta>

    <title>SIG construido con GeoExt y OpenLayers</title>

    <!-- Estilos-->
    <link rel="stylesheet" type="text/css" href="lib/ext/resources/css/ext-all.css" />
    <link rel="stylesheet" type="text/css" href="lib/GeoExt/resources/css/popup.css" />

    <!-- Frameworks y librerías-->
    <script type="text/javascript" src="lib/ext/adaptor/ext/ext-base.js"></script>
    <script type="text/javascript" src="lib/ext/ext-all.js"></script>
    <script src="lib/OpenLayers-2.10/OpenLayers.js"></script>
    <script type="text/javascript" src="lib/GeoExt/script/GeoExt.js"></script>

    <!-- Clases propias-->
    <script type="text/javascript" src="lib/edu/eafit/campus/pack1/main.js"></script>
  </head>
  <body>
  </body>
</html>
```

Figura 47. Modularización básica

[2] Tres ejemplos de encapsulación en JavaScript se muestran a continuación:

- Sencilla, con *OpenLayers*:

```
/**
 * Hereda de:
 * - <OpenLayers.Control>
 */
OpenLayers.Control.DeleteFeature = OpenLayers.Class(OpenLayers.Control, {

  /**
   * Constructor: OpenLayers.Control.DeleteFeature
   */
  initialize: function(layer, options) {
    OpenLayers.Control.prototype.initialize.apply(this, [options]);
    this.layer = layer;
    this.handler = new OpenLayers.Handler.Feature(
      this, layer, {click: this.clickFeature}
    );
  },
  clickFeature: function(feature) {
    this.layer.destroyFeatures([feature]);
  },
  CLASS_NAME: "OpenLayers.Control.DeleteFeature"
});
```

Figura 48. Encapsulación en OL

- Avanzada con *ExtJS (v3.4)*:

```

Ext.ns("com.eafit.app");
(function(){
    var JSClass = Ext.extend(Object,{

        constructor: function(config){
            Ext.apply(this, config)
        },
        //atributos
        attA:null,
        attB:null,

        //métodos
        getAttA: function(){
            return attA;
        },

        setAttA: function(att){
            this.attA = att;
        }
    })
    com.eafit.app.JSClass = JSClass;
})();

```

Figura 49. Encapsulación en *ExtJS*

- Avanzada, con *Dojo (v1.6)*:

```

dojo.declare('Logger', null, {

    constructor: function (outputId) {
        this.node= dojo.byId(outputId)
    },

    log: function(text) {
        if(text==undefined)
            return
        this.node.innerHTML+="  
>" + text
    }

})

```

Figura 50. Encapsulación en *Dojo*

[3] Para una aplicación tipo avanzada, se requiere de un patrón arquitectónico, que facilite la creación de la IU, la distribución de tareas entre el equipo de desarrollo y la integración. Un patrón del tipo *Model View Controller* para las versiones ExtJS 3.4 y Dojo 1.6 se puede implementar parcialmente.

- Para el caso de *GeoExt*, se articula un patrón *View Model*, en el cual hay una clase que además de ser responsable de la IU, también controla el llamado a los otros componentes o clases del modelo, que tienen a su cargo el resto de las funcionalidades. Un ejemplo de clase, denominada *ViewLayout* se muestra a continuación:

```

Ext.ns("edu.eafit.campus.pack2");

(function(){
    var ViewLayout = Ext.extend(Object,{
        constructor:function(config){
            Ext.apply(this,config);
        },

        createLayout:function(){
            //Paneles

            var northPanel = new Ext.Panel({
                id:'north-panel', height:80, region:'north',border:false, title:'Encabezad
            });

            var southPanel = new Ext.Panel({
                id:'south-panel', height:200, region:'south', title:'M&#225;s informaci&#225;n', c
            });

            var map=new edu.eafit.campus.pack2.WMSCampus();
            map.createMapa();
            var blq=new edu.eafit.campus.pack3.BloquesWFS();
            blq.createBlq();
            var locallyBlq=blq.getlyBlq();

            var centerPanel =new GeoExt.MapPanel({
                id: "mappanel",title: "Mapa",region:'center', layout:'fit',
                map:map.getMapa(), layers:[locallyBlq]
            });

            var westPanel = new Ext.Panel({
                id:'west-panel', title: 'Men&#225;o', layout:'accordion',region:'west',width:300
            });

            new Ext.Viewport({
                id:'id-viewport',
                layout:'border',
                items: [northPanel,southPanel,westPanel,centerPanel]
            });
        }
    });
    edu.eafit.campus.pack2.ViewLayout=ViewLayout;
})();

```

Figura 51. Patrón View Model

- Para el caso del API JavaScript de ArcGIS Server, se puede implementar una clase controladora o *Application Controller*, que tiene la responsabilidad del manejo de la aplicación, asume el control del ciclo de vida y la carga de sus partes (las inicializa y conecta). Conoce los componentes gráfcos que están en la página HTML y hace los llamados a las clases del modelo. Un ejemplo de clase, llamada *base2*, se muestra a continuacián:

```

dojo.provide('app.base2');

app.base2 = {
  mapa : null,
  init : function() {

    this.startup();
    dojo.require("app.module");

  },
  startup : function() {

    this.mapa = new app.mapa();
    this.mapa.crearMapaBase();
    this.initUi();

  },
  initUi : function() {

    dojo.connect(dojo.byId("idenBtn"), "onclick", this, function(evt) {
      dojo.connect(this.mapa.getMapa(), "onClick", this, function() {
        this.mapa.getMapa().graphics.clear();
        this.doSearchIdenti(evt);
      });
    });
    dojo.connect(dojo.byId("searchBtn"), "onclick", this, function() {
      this.doSearchText(dojo.byId('searchText').value);
    });

  },
  doSearchIdenti : function(evt) {
    identif = new app.identificar2();
    identif.doIdentificar(this.mapa.getMapa(), evt);
  },
  doSearchText : function(texto) {
    busq2 = new app.busqueda2();
    busq2.doBuscar(this.mapa.getMapa(), texto);
  }
};

```

Figura 52. Patrón *Application Controller*

3.2 DURANTE EL DESARROLLO DEL PROYECTO

- Mantener siempre presente el tipo de usuario a quien va dirigida la aplicación y diferenciarlo del rol del cliente, quien solicita el desarrollo del producto y tiene mayor conocimiento de los requisitos (i.e. entidad de gobierno como alcaldías municipales, secretarías), mientras que el usuario es la persona que interactúa directamente con la aplicación (i.e. ciudadano corriente). El contenido y los mapas que se brindan deben ser útiles e interesantes, de manera que cautiven la atención del usuario. El éxito de una aplicación además de estar mediada por el cumplimiento de los requisitos, está relacionada con el provecho que las personas saquen de ella, y en algunos casos por el crecimiento en número de usuarios. La utilización de herramientas disponibles en línea (el caso de ArcGIS Online) es valiosa para hacer demostraciones al cliente y hacer acercamientos con el usuario, que si bien no van a ser parte de la solución final sirven para retroalimentar el desarrollo. Las siguientes imágenes refieren a tres principios a ser tomados en cuenta: en primer lugar, KISS (*keep it simple*), usabilidad y rendimiento- velocidad de respuesta



Figura 53. KISS, usabilidad y rendimiento. Tomadas de (Anta, 2012) (izq) y (Lamas, 2011) (der y abajo)

- Priorizar las funcionalidades a implementar: aquellas que tengan valor significativo para el cliente deben ocupar los primeros lugares.
- Analizar qué calidades sistémicas prevalecerán sobre otras, ya que esto impacta directamente a la arquitectura, por ejemplo, si se requiere de altos niveles de seguridad, el *performance* se ve afectado; si la disponibilidad es de 7 X 24, se deben contar con mecanismos de redundancia; si el público a quien se dirige la aplicación es no experto, la usabilidad y la experiencia de usuario deben primar.
- Conformar un grupo para la selección de tecnologías, entre los integrantes deben figurar el director y el arquitecto. Las condiciones del proyecto, la experiencia y las pruebas de concepto son determinantes en este proceso.
- El punto de partida para la aplicación es generar un esquema de GUI (Interfaz gráfica de usuario), por ejemplo, la siguiente imagen muestra un *layout* tradicional:

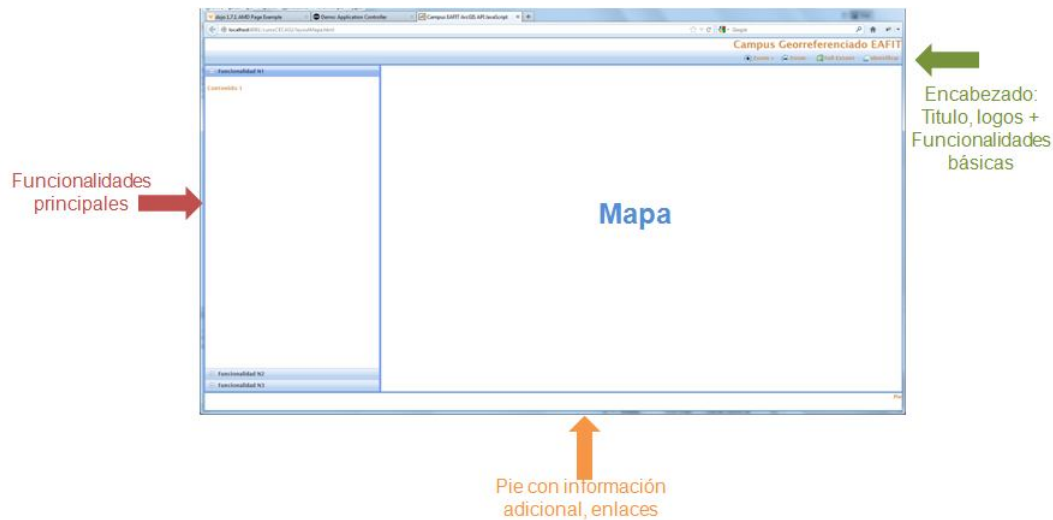


Figura 54. Layout básico

El mapa es el objeto más importante dentro la aplicación, por ello ocupa buena parte de la página y está centrado. Sobre él se efectuarán la gran mayoría de operaciones, entonces es necesario crear una clase que lo represente y pueda ser compartida por todas las funcionalidades, el concepto se asemeja a un *singleton* del cual dependen los demás.

Las funcionalidades en el *layout* presentado, se ubican a la izquierda, se asignan a los desarrolladores, permitiendo la codificación y depuración de forma independiente.

Debe existir un controlador que interprete las acciones del usuario y pueda llamar a la clase responsable de ejecutar la acción determinada. También se encarga de la inicialización y la conexión de las partes en la secuencia correcta.

Esta arquitectura base, en la cual se hace descomposición modular facilitará en gran medida el proceso de integración posterior. Dependiendo del framework se pueden aplicar otros patrones en la construcción de la aplicación.

- Configurar el ambiente de desarrollo, de forma que sea un estándar para todo el equipo de trabajo. Algunas herramientas que pueden emplearse son: aptana, notepad++ o Microsoft Visual Web Developer Express. De igual, forma es indispensable contar con un sistema de control de versiones (CVS).
- Dado que la información que se brinde a través del mapa debe estar vigente y ser de interés, se debe generar un plan para la actualización de datos, que contemple la frecuencia, la entidad responsable (empresa desarrolladora, cliente o compartida) y costos aproximados. Por ejemplo: en una aplicación de geomercado se pueden establecer convenios con proveedores especializados para adquirir el levantamiento de información de establecimientos comerciales que se realiza cada dos años.

- Y aunque este último lineamiento no pertenezca al campo técnico, vale mencionar que buena parte del éxito de un proyecto recae sobre las personas, entonces es vital que se incentive un ambiente de buenas relaciones, de cooperación en el que se estimule el trabajo en equipo.

Los siguientes lineamientos se dirigen hacia los desarrolladores:

- Suscribirse a foros y listas de correo es una buena opción tanto para buscar ayuda como para colaborar con otros programadores, tal es el caso de lists.osgeo.org.
- Iniciar buscando los ejemplos con funcionalidad similar a la deseada, si no se encuentra, pasar al trac y/o a la lista de correo y una vez que se tenga una idea de la línea a tomar apoyarse en la documentación del API.
- Es fundamental que para las tareas de depuración cuente con una herramienta, dado que son aplicaciones del lado cliente, el navegador sobre el que está probando debe incluirla. Por ejemplo: Firebug para Firefox, Dragonfly para Opera, el inspector de Google Chrome. Es útil tanto para ubicar errores como para hacer pruebas de rendimiento, al medir cuánto tiempo se demora una función (o trozo de código).
- Hacer de las buenas prácticas una rutina diaria, de manera que se interioricen y se conviertan en hábitos, por ejemplo: evitar variables globales y anidamientos excesivos; optimizar bucles; asignar nombres - ojalá cortos- a las variables y funciones que se auto expliquen; los comentarios deben ser los necesarios; utilizar un analizador de código para comprobar la calidad sintáctica, esto permite identificar problemas potenciales (i.e. faltas de punto y coma, variables globales, defectos en el HTML), ahorrando tiempo en el proceso de depuración y mejorando la integración con los productos de otros desarrolladores (entre las herramientas disponibles está JSLint).
- La minimización y *obfuscation* de código son alternativas que se deben tener presentes para optimizar tiempos de carga.

3.3 DESPUÉS DEL PROYECTO

Estos lineamientos conducen a la evaluación y sostenibilidad técnica del proyecto, donde a partir de la experiencia y aprendizaje, se establezcan diagnósticos y se propongan opciones de cambio y mejora para próximos proyectos.

Estos lineamientos se encuentran enfocados desde tres puntos:

El software creado en sí mismo:

- Grado de reutilización: cuántos módulos pueden emplearse en construir otra aplicación (con características similares).

- Cuál es el componente que es más susceptible a presentar fallos, para proponer alternativas que mitiguen los riesgos.
- Cuantificar tiempos de respuesta, en especial cuando hay picos de peticiones, se debe considerar tanto del *backend* como el *frontend*.

Respecto al cliente / usuario:

- Medir el grado de satisfacción del cliente: donde éste califique cuantitativa y cualitativamente el trabajo realizado, en aspectos que van desde el cumplimiento de tiempos, receptividad ante adiciones y cambios propuestos en las funcionalidades, metodología empleada, entre otros.
- Medir el grado de utilización por parte de los usuarios: saber cuántas personas ingresan a la aplicación (contador de visitas); poner a disposición en la misma página una encuesta breve sobre si le gusta y un espacio para que manifiesten ideas de ampliar la funcionalidad o para presentar inconformidades sobre el aplicativo. Esta perspectiva permite obtener un panorama de la evolución del software.

Desde el equipo de desarrollo:

- Obtener y analizar percepciones sobre la metodología de desarrollo empleada, de modo que se propongan ajustes y se apliquen para los proyectos siguientes.
- Grado de dominio de tecnologías, curva de aprendizaje, preguntar si en un próximo proyecto se emplearían las conocidas o se optarían por otras (i.e en lugar de Dojo utilizar jQuery).
- Examinar si el ambiente de desarrollo, el sistema de control de versiones y demás herramientas fueron efectivos y eficaces. Listar posibles tecnologías sustitutivas y complementarias.

CONCLUSIONES Y TRABAJOS FUTUROS

Los lineamientos parten de una base teórica, recogen y abstraen las lecciones derivadas de un proceso práctico, independizándolos de herramientas o plataformas particulares (agnósticos), de modo que pueden utilizarse en otro tiempo y circunstancias. Sin embargo, las tecnologías por su rápida y constante evolución pueden desembocar cambios que modifiquen o supriman algunos lineamientos de los propuestos, lo cual les otorga una naturaleza dinámica.

Las aplicaciones piloto constituyeron el medio y no el fin en el proceso de estructurar los lineamientos, su construcción fue una buena forma de vivenciar el proceso de desarrollo de software, con los inconvenientes en cada una de las etapas, desde los requisitos hasta la implementación, permitiendo tener varios panoramas, desde la visión a gran escala de la arquitectura hasta los detalles de la codificación.

Toda herramienta tecnológica (aplicación, plataforma, *framework*) tiene pros y contras, lo importante es hacer el balance y determinar cuál de todas las que se encuentran disponibles es la que mejor se adapta a las necesidades y brinda más y mejores beneficios a un proyecto en particular. En muchos casos es posible utilizar híbridos entre tecnologías libres y propietarias, debido al uso de servicios y a la estandarización, tal como se demostró en la construcción de las aplicaciones piloto.

Si bien las tecnologías juegan un papel determinante en minimizar riesgos y aumentar las posibilidades de éxito de una aplicación SIG Web, también son pilares la conformación del equipo de desarrollo y la metodología de trabajo.

Se propone conformar un banco de conocimiento o hacer gestión de conocimiento que recoja las experiencias de desarrolladores de SIG Web y tecnologías móviles, campo que se encuentra en pleno auge y expansión. En especial, en el mundo del software libre este tipo de iniciativas tienden a tener buena acogida.

ANEXO A. CODIGO FUENTE DE LAS APLICACIONES PILOTO

Apartes del código del piloto construido con software libre

proxy.cgi

Debido a restricciones de seguridad en JavaScript, no es posible recuperar información de dominios remotos a través de XMLHttpRequest, por esto es necesario instalar un proxy. Se modificó el propuesto por OpenLayers.

```
12 import urllib2
13 import cgi
14 import sys, os
15
16 # Designed to prevent Open Proxy type stuff.
17
18 allowedHosts = ['localhost:8080',
19                'tah.openstreetmap.org']
20
21 method = os.environ["REQUEST_METHOD"]
22
23 if method == "POST":
24     qs = os.environ["QUERY_STRING"]
25     d = cgi.parse_qs(qs)
26     if d.has_key("url"):
27         url = d["url"][0]
28     else:
29         url = "http://www.openlayers.org"
30 else:
31     fs = cgi.FieldStorage()
32     url = fs.getvalue('url', "http://www.openlayers.org")
33
34 try:
35     host = url.split("/")[2]
36     if allowedHosts and not host in allowedHosts:
37         print "Status: 502 Bad Gateway"
38         print "Content-Type: text/plain"
39         print
40         print "This proxy does not allow you to access that location (%s)." % (host,)
41         print
42         print os.environ
43
44 elif url.startswith("http://") or url.startswith("https://"):
45
46     if method == "POST":
47         length = int(os.environ["CONTENT_LENGTH"])
48         headers = {"Content-Type": os.environ["CONTENT_TYPE"]}
49         body = sys.stdin.read(length)
50         r = urllib2.Request(url, body, headers)
51         y = urllib2.urlopen(r)
52     else:
53         y = urllib2.urlopen(url)
54
55     # print content type header
56     i = y.info()
57     if i.has_key("Content-Type"):
58         print "Content-Type: %s" % (i["Content-Type"])
59     else:
60         print "Content-Type: text/plain"
61     print
62
63     print y.read()
64
65     y.close()
66 else:
67     print "Content-Type: text/plain"
68     print
69     print "Illegal request."
70
71 except Exception, E:
72     print "Status: 500 Unexpected Error"
73     print "Content-Type: text/plain"
74     print
75     print "Some unexpected error occurred. Error text was:", E
76
```

main.js

Constituye el punto de entrada de la aplicación.

```
1 Ext.BLANK_IMAGE_URL = 'lib/ext/resources/images/default/s.gif';
2
3 // Punto de entrada principal de la aplicación
4 Ext.onReady(function() {
5
6     Ext.QuickTips.init();
7
8     var onModuleLoad=function() {
9         var viewLayout=new edu.eafit.campus.pack2.ViewLayout();
10        viewLayout.createLayout();
11    }
12
13    onModuleLoad();
14 });
```

ViewLayout.js

Realiza la división o disposición del espacio en regiones (*north*, *south*, *west*, *east*, *center*) a través de paneles.

```
1 Ext.ns("edu.eafit.campus.pack2"); (function() {
2     var ViewLayout = Ext.extend(Object, {
3         constructor : function(config) {
4             Ext.apply(this, config);
5         },
6         //creación del layout de la página
7
8         createLayout : function() {
9             //Paneles
10
11             var northPanel = new Ext.Panel({
12                 id : 'north-panel',
13                 height : 80,
14                 region : 'north',
15                 border : false,
16                 title : 'Encabezado para imagen'
17             });
18
19             var southPanel = new Ext.Panel({
20                 id : 'south-panel',
21                 height : 200,
22                 region : 'south',
23                 title : 'Más información',
24                 collapsible : true,
25                 collapsed : true
26             });
27
28             var map = new edu.eafit.campus.pack2.WMSCampus();
29             map.createMapa();
30
31             var blq = new edu.eafit.campus.pack3.BloquesWFS();
32             blq.createBlq();
33
34             var ptoIn = new edu.eafit.campus.pack3.PuntosIWFS();
35             ptoIn.createPtosIn();
36
37             var locallyPtosIn = ptoIn.getlyPtosIn();
38
39             var locallyBlq = blq.getlyBlq();
40
41             var ctrl, toolbarItems = [], action, actions = {};
42
43             // ZoomToMaxExtent control, a "button" control
44             action = new GeoExt.Action({
45                 control : new OpenLayers.Control.ZoomToMaxExtent(),
46                 map : map.getMapa(),
47                 text : "max extent",
48                 tooltip : "zoom to max extent"
```

```
49     });
50     actions["max_extent"] = action;
51     toolbarItems.push(action);
52     toolbarItems.push("-");
```

```
109
110 var info = new OpenLayers.Control.WMSGetFeatureInfo({
111     layers : [map.getMapa().layers[2], map.getMapa().layers[3]],
112     title : 'Identify features by clicking',
113     infoFormat : 'application/vnd.ogc.gml',
114     queryVisible : true,
115     eventListeners : {
116         getfeatureinfo : function(e) {
117             var html = '<table>';
118             var features = this.format.read(e.text);
119             // var features = e.features;
120             console.log('features crudas' + features.length);
121             if(features && features.length > 0) {
122                 console.log('features procesadas' + features[0].attributes.BLQ_NOMBRE);
123                 //console.log('features procesadas'+features[0].attributes[1]);
124                 html += '<tr><th></th><td>';
125                 html += '<tr><th>' + features[0].attributes.BLQ_NOMBRE + '</th><td>';
126                 html += '</table>';
127                 new GeoExt.Popup({
128                     title : "Feature Info",
129                     width : 250,
130                     height : 150,
131                     unpinable : false,
132                     collapse : false,
133                     collapsible : false,
134                     resizable : false,
135                     anchored : true,
136                     map : map.getMapa(),
137                     location : new OpenLayers.LonLat(-8413398.0571968, 691453.20375893), //-8413398.0571968,lat=691453.20375893
138                     //location: map.getMapa().getLonLatFromPixel(e.xy),
139                     html : html
140                 }).show();
141             }
142         }
143     }
144 });
145
146 action = new GeoExt.Action({
147     control : info,
148     map : map.getMapa(),
149     text : "identificar",
150     tooltip : "hacer click en el mapa"
151 });
152 actions["identificar"] = action;
153 toolbarItems.push(action);
154 toolbarItems.push("-");
155
156 map.getMapa().addControl(info);
```



```

158 info.activate();
159
160 var centerPanel = new GeoExt.MapPanel({
161     id : "mappanel",
162     title : "Mapa",
163     region : 'center',
164     layout : 'fit',
165     xtype : "gx_mappanel",
166     map : map.getMapa(),
167     layers : [locallyBlq, locallyPtosIn],
168     tbar : toolbarItems
169 });
170
171 var item1 = new Ext.FormPanel({
172     title : 'Oficinas - Aulas',
173     frame : true,
174     border : false,
175     items : [{
176         xtype : 'textfield',
177         name : 'txt-ofi',
178         emptyText : 'Por ejemplo: 18505',
179         width : 125
180     }],
181     buttons : [{
182         text : 'Buscar',
183         handler : function() {
184
185             var nombre = item1.getForm().findField('txt-ofi').getValue();
186             var ofi = new edu.eafit.campus.pack3.OfiWFS();
187
188             ofi.createOf(nombre.substring(0, 2), nombre.substring(2, 3), nombre);
189
190             var oflocally = ofi.getlyOf();
191             centerPanel.map.addLayer(oflocally);
192             console.log("Total coorde:" + ofi.getcoorOf().x);
193
194             centerPanel.map.setCenter(new OpenLayers.LonLat(ofi.getcoorOf().x, ofi.getcoorOf().y), 40);
195
196             var popup = new edu.eafit.campus.pack2.PopUpCampus();
197
198             popup.createPopUp(new OpenLayers.LonLat(ofi.getcoorOf().x, ofi.getcoorOf().y), centerPanel.map, ofi.getstoreOf());
199
200         }
201     }],
202     keys : [{
203         key : [Ext.EventObject.ENTER],
204         handler : function() {
205
206             var nombre = item1.getForm().findField('txt-ofi').getValue();
207             var ofi = new edu.eafit.campus.pack3.OfiWFS();
208             ofi.createOf(nombre.substring(0, 2), nombre.substring(2, 3), nombre);
209
210             var oflocally = ofi.getlyOf();
211             centerPanel.map.addLayer(oflocally);
212
213         }
214     }
215     });
216
217 var item2 = new Ext.grid.GridPanel({
218     title : "Bloques",
219     frame : true,
220     border : false,
221     viewConfig : {
222         forceFit : true
223     },
224     store : blq.getStBlq(),
225     sm : new GeoExt.grid.FeatureSelectionModel(),
226     cm : new Ext.grid.ColumnModel({
227         defaults : {
228             sortable : true
229         },
230         columns : [{
231             header : "",
232             dataIndex : "BLQ_NOMBRE"
233         }, {
234             header : "#",
235             dataIndex : "BLQ_NOMB_1"
236         }
237     ]
238     });
239
240 var item3 = new Ext.grid.GridPanel({
241     title : "Puntos de interés",
242     frame : true,
243     border : false,
244     viewConfig : {
245         forceFit : true
246     },
247     store : ptoIn.getStPtosIn(),
248     sm : new GeoExt.grid.FeatureSelectionModel(),
249     cm : new Ext.grid.ColumnModel({
250         defaults : {
251             sortable : true
252         },

```

```

253     columns : [{
254         header : "Sitio",
255         dataIndex : "PIN_NOMBRE"
256     }]
257     })
258 });
259
260 var item4 = new Ext.tree.TreePanel({
261     title : "Capas del mapa",
262     autoScroll : true,
263     enableDD : true,
264     lines : false,
265     rootVisible : false,
266     root : new GeoExt.tree.LayerContainer({
267         layerStore : centerPanel.layers,
268         expanded : true
269     }),
270     bbar : [{
271         text : "Remove from Map",
272         handler : function() {
273             var node = tree.getSelectionModel().getSelectedNode();
274             if(node) {
275                 centerPanel.map.removeLayer(node.layer);
276             }
277         }
278     }]
279 });
280
281 var westPanel = new Ext.Panel({
282     id : 'west-panel',
283     title : 'Menú',
284     layout : 'accordion',
285     region : 'west',
286     width : 300,
287     split : true,
288     layoutConfig : {
289         // layout-specific configs go here
290         animate : true
291     },
292     items : [item1, item2, item3, item4]
293 });
294
295 //Viewport
296
297 new Ext.Viewport({
298     id : 'id-viewport',
299     layout : 'border',
300
301     items : [northPanel, southPanel, westPanel, centerPanel]
302 });
303 }
304 });
305 });
306
307 edu.eafit.campus.pack2.ViewLayout = ViewLayout;
308 }) ();

```

WMSCampus.js

Se encarga de traer las capas de fondo del mapa, tanto las de Google como las del Campus, haciendo peticiones WMS.

```
1 Ext.ns("edu.eafit.campus.pack2"); (function() {
2   var WMSCampus = Ext.extend(Object, {
3     constructor : function(config) {
4       Ext.apply(this, config);
5     },
6     mapBM : null,
7     createMapa : function() {
8       //console.log("en el mapa");
9
10      var epsg900913 = new OpenLayers.Projection("EPSG:900913");
11      var epsg4326 = new OpenLayers.Projection("EPSG:4326");
12      var extent = new OpenLayers.Bounds(-75.58, 6.197, -75.576, 6.203);
13
14      var options = {
15        projection : epsg900913,
16        displayProjection : epsg4326,
17        units : "m",
18        numZoomLevels : 40,
19        maxResolution : 156543.0339,
20        maxExtent : extent
21      };
22
23      extent.transform(new OpenLayers.Projection("EPSG:4326"), options.projection);
24
25      var base = new OpenLayers.Layer.Google("Google Streets", {
26        'sphericalMercator' : true,
27        numZoomLevels : 40,
28        maxExtent : extent
29      });
30
31      var base_sat = new OpenLayers.Layer.Google("Google Satellite", {
32        type : G_SATELLITE_MAP,
33        sphericalMercator : true,
34        numZoomLevels : 40,
35        maxExtent : extent
36      });
37
38      var bloques_eafit = new OpenLayers.Layer.WMS("Bloques EAFIT", "http://localhost:8080/geoserver/ora/wms", {
39        layers : "ora:CMFC_BLOQUES_TB",
40        transparent : true
41      }, {
42        isBaseLayer : false,
43        visibility : true,
44        buffer : 0,
45        opacity : 0.9
46      });
47
48
49      var esp_abier = new OpenLayers.Layer.WMS("Espacios EAFIT", "http://localhost:8080/geoserver/ora/wms", {
50        layers : "ora:CMFC_ESPACIO_ABIERTO_TB",
51        transparent : true
52      }, {
53        isBaseLayer : false,
54        visibility : true,
55        buffer : 0,
56        opacity : 0.1
57      });
58
59      mapBM = new OpenLayers.Map();
60      mapBM = new OpenLayers.Map(options);
61      var layers = [base_sat, base, bloques_eafit, esp_abier];
62
63      mapBM.addLayers(layers);
64
65      console.log("layersI " + mapBM.getNumLayers());
66
67    },
68    getMapa : function() {
69      return mapBM;
70    },
71    setMapa : function(mp) {
72      this.mapBM = mp;
73    }
74  })
75
76  edu.eafit.campus.pack2.WMSCampus = WMSCampus;
77 })();
```

BloquesWFS.js

Corresponde a una clase del modelo, que se encarga de traer los datos del WFS, en este caso de Bloques.

```
1 Ext.ns("edu.eafit.campus.pack3"); (function() {
2   var BloquesWFS = Ext.extend(Object, {
3     //función constructora, que se ejecutará cuando se cree la clase.
4     constructor : function(config) {
5       //console.log("crear bloques");
6       Ext.apply(this, config);
7     },
8     //atributos
9
10    storeBlq : null,
11    layerBlq : null
12
13    //funciones
14
15
16    ,
17    createBlq : function() {
18
19      console.log("en WFS");
20
21      OpenLayers.ProxyHost = "cgi-bin/proxy.cgi?url=";
22
23      var epsg4326 = new OpenLayers.Projection("EPSG:4326");
24
25      //Recomendación: crear una capa tipo vector con protocolo WFS
26
27      var estiloBWFS = new OpenLayers.StyleMap({
28
29        "default" : new OpenLayers.Style(null, {
30          rules : [new OpenLayers.Rule({
31
32            symbolizer : {
33              "Polygon" : {
34                graphicName : "pol",
35                fillColor : "#FF0080",
36                fillOpacity : 0,
37                strokeWidth : 0,
38                strokeOpacity : 0,
39                strokeColor : "#8A084B"
40              }
41            }
42          }
43        })]
44      });
45      layerBlq = new OpenLayers.Layer.Vector("BloquesWFS", {
46        strategies : [new OpenLayers.Strategy.Fixed()],
47        projection : new OpenLayers.Projection(epsg4326),
48        styleMap : estiloBWFS,
49        protocol : new OpenLayers.Protocol.WFS({
```

```
49         url : "http://localhost:8080/geoserver/wfs",
50         version : "1.1.0",
51         featureType : "GMFC_BLOQUE3_TB",
52         featureNS : "ora",
53         extractAttributes : true
54     })
55
56     });
57     storeBlq = new GeoExt.data.FeatureStore({
58     fields : [{
59         name : "BLQ_PK_NUM",
60         type : "string"
61     }, {
62         name : "BLQ_NOMBRE",
63         type : "string"
64     }, {
65         name : "BLQ_NOME_1",
66         type : "string"
67     }],
68     layer : layerBlq
69     });
70
71     },
72     getStBlq : function() {
73         return storeBlq;
74     },
75     setStBlq : function(bc) {
76         this.storeBlq = bc;
77     },
78     getlyBlq : function() {
79         return layerBlq;
80     },
81     setlyBlq : function(lb) {
82         this.layerBlq = lb;
83     },
84     } // fin definición de la clase.
85
86     edu.eafit.campus.pack3.BloquesWFS = BloquesWFS;
87     } ();
```

Apartes del código del piloto construido con software propietario

indexAC2.html

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
5 <meta http-equiv="X-UA-Compatible" content="IE=7" />
6 <!--The viewport meta tag is used to improve the presentation and behavior of the samples
7 on iOS devices-->
8 <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no">
9 <title></title>
10 <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.7/js/dojo/dijit/themes/soria/soria.css">
11 <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.7/js/dojo/dojo/grid/resources/Grid.css">
12 <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.7/js/dojo/dojo/grid/resources/SoriaGrid.css">
13 <link rel="stylesheet" type="text/css" href="http://serverapi.arcgisonline.com/jsapi/arcgis/2.7/js/esri/dijit/css/Popup.css">
14 <link rel="stylesheet" href="http://AppDojo/css/demo.css" media="screen">
15 <link rel="stylesheet" href="http://AppDojo/css/style.css" media="screen">
16 <script type="text/javascript">
17     var djConfig = {
18         parseOnLoad : true,
19         baseUrl : './',
20         modulePaths : {
21             app : 'js/app',
22             'agsjs' : 'js/agsjs'
23         }
24     };
25 </script>
26 <script type="text/javascript" src="http://serverapi.arcgisonline.com/jsapi/arcgis/?v=2.7"></script>
27 <script>
28     dojo.require("app.base2");
29
30     dojo.ready(function() {
31         app.base2.init();
32     });
33 </script>
34 </head>
35 <body class="soria">
36
37     <div id="mainWindow" dojotype="dijit.layout.BorderContainer" design="headline" gutters="false" style="width:100%; height:100%;">
38     <div id="header" dojotype="dijit.layout.ContentPane" class="roundedCorners" region="top" >
39         Campus Georreferenciado EAFIT
40         <div id="navToolBar" data-dojotype="dijit.Toolbar">
41             <div data-dojotype="dijit.form.Button" id="zoomin" data-dojotype="iconClass:'zoominIcon', onClick:function(){navToolBar.activate(esri.toolbars.Navigation.ZOOM_I
42                 Zoom +
43             </div>
44             <div data-dojotype="dijit.form.Button" id="zoomout" data-dojotype="iconClass:'zoomoutIcon', onClick:function(){navToolBar.activate(esri.toolbars.Navigation.ZOOM_
45                 Zoom -
46             </div>
47             <div data-dojotype="dijit.form.Button" id="zoomfullExt" data-dojotype="iconClass:'zoomfullExtIcon', onClick:function(){navToolBar.zoomToFullExtent();}>
48
49                 Full Extent
50             </div>
51             <div data-dojotype="dijit.form.Button" id="idenBtn" data-dojotype="iconClass:'identificarIcon '">
52                 Identificar</button>
53             </div>
54         </div>
55     </div>
56     <div dojotype="dijit.layout.ContentPane" class="roundedCorners" region="left" id="leftPane" >
57         <div dojotype="dijit.layout.AccordionContainer">
58             <div dojotype="dijit.layout.ContentPane" title="Búsqueda espacios" splitter="true" >
59                 <input type="text" id="searchText" value="B 18" />
60                 <button dojotype="dijit.form.Button" id="searchBtn">
61                     Buscar
62                 </button>
63                 <table dojotype="dojo.grid.DataGrid" jsid="grid" id="grid" style="width:100%;height:100%;selectionMode="none">
64                     <thead>
65                         <tr>
66                             <th field="OBJECTID" width="15px">ID</th>
67                             <th field="BLQ_NOMBRE">Nombre</th>
68                             <th field="BLQ_NOMBRE">Sitio</th>
69                         </tr>
70                     </thead>
71                 </table>
72             </div>
73             <div dojotype="dijit.layout.ContentPane" title="Ruta" splitter="true">
74                 <input type="text" id="ruta" value="ruta 1" />
75                 <button dojotype="dijit.form.Button" id="rutaBtn">
76                     Ruta
77                 </button>
78             </div>
79             <div dojotype="dijit.layout.ContentPane" title="Búsqueda 3">
80                 <p>
81                     Contenido 3
82                 </p>
83             </div>
84         </div>
85     </div>
86     <div id="mapDiv" class="roundedCorners" dojotype="dijit.layout.ContentPane" region="center" ></div>
87     <div id="footer" class="roundedCorners" dojotype="dijit.layout.ContentPane" region="bottom" >
88         Pie
89     </div>
90 </div>
91 </body>
92 </html>
93
```

base2.js

Esta clase conforma el *application controller*, inicializa y conecta las partes en la secuencia correcta.

```
1  dojo.provide('app.base2');
2
3  app.base2 = {
4    mapa : null,
5
6    init : function() {
7      // proceed directly with startup
8      this.startup();
9      dojo.require("app.module");
10   },
11   startup : function() {
12     // create the data store
13     /* var objMap = new app.map();
14     objMap.crearionMapaBase(); */
15
16     this.mapa = new app.mapa();
17     this.mapa.crearMapaBase();
18     this.initUi();
19   },
20   initUi : function() {
21
22     dojo.connect(dojo.byId("idenBtn"), "onclick", this, function(evt) {
23       dojo.connect(this.mapa.getMapa(), "onClick", this, function(evt) {
24         this.mapa.getMapa().graphics.clear();
25         this.doSearchIdent1(evt);
26       });
27     });
28     dojo.connect(dojo.byId("searchBtn"), "onclick", this, function(evt) {
29       this.doSearchText(dojo.byId('searchText').value);
30     });
31     /* dojo.connect(grid, "onMouseOverRow", this, function(evt){
32     if(evt.rowIndex === -1){
33       return;
34     }
35     this.doSyncGrid(grid, evt.rowIndex);
36     }); */
37
38     dojo.connect(grid, "onMouseOverRow", function(evt) {
39       if(evt.rowIndex === -1) {
40         return;
41       }
42       selectionLayer = new esri.layers.GraphicsLayer();
43       selectionLayer.setRenderer(new esri.renderer.SimpleRenderer(new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 255, 0, 0.50]))));
44       this.mapa.getMapa().addLayer(selectionLayer);
45
46       var defaultRenderer = new esri.renderer.SimpleRenderer(new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([0, 0, 0])));
47       this.mapa.getMapa().graphics.setRenderer(defaultRenderer);
48       var rowId = grid.getItem(evt.rowIndex).OBJECTID;
```

```

49     selectionLayer.clear();
50     //add a new graphic to the selection layer
51     //dojo.some(this.mapa.getMapa().graphics.graphics,function(graphic){
52     dojo.some(this.mapa.getMapa().graphics, function(graphic) {
53         if((graphic.attributes) && graphic.attributes.OBJECTID === rowId) {
54             var selectedState = new esri.Graphic(graphic.geometry).setAttributes(graphic.attributes);
55             selectionLayer.add(selectedState);
56             return true;
57         }
58     });
59     });
60
61     dojo.connect(grid, "onStyleRow", function(row) {
62         if(row.over) {
63             row.customStyles += 'background-color:#FFFF00;';
64         }
65     });
66
67     dojo.connect(grid, "onMouseOutRow", function() {
68         selectionLayer.clear();
69     });
70
71     dojo.connect(this.mapa.getMapa().graphics, "onMouseOver", function(evt) {
72         selectionLayer.clear();
73         //loop through the grid and find the row associated with the graphic
74         for(var i = 0, il = grid.rowCount; i < il; i++) {
75             if((grid.getItem(i) && grid.getItem(i).OBJECTID === evt.graphic.attributes.OBJECTID) {
76                 grid.rows.setOverRow(i);
77                 grid.scrollToRow(i);
78                 var selectedState = new esri.Graphic(evt.graphic.geometry).setAttributes(evt.graphic.attributes);
79                 selectionLayer.add(selectedState);
80                 break;
81             }
82         }
83     });
84
85     dojo.connect(this.mapa.getMapa().graphics, "onMouseOut", function(evt) {
86         // selectionLayer.clear();
87         grid.rows.setOverRow(-1);
88     });
89
90     dojo.connect(dojo.byId("rutaBtn"), "onclick", this, function(evt) {
91         this.doGP(dojo.byId('ruta').value);
92     });
93
94     doSearchIdentif : function(evt) {
95         //identif= new app.identificar();
96         identif = new app.identificar2();

```

```

97         identif.doIdentificar(this.mapa.getMapa(), evt);
98     },
99     doSearchText : function(texto) {
100         console.log("tengo esto " + texto);
101         /*busq= new app.búsqueda();
102         busq.doBuscar(this.mapa.getMapa(),texto);*/
103         busq2 = new app.búsqueda2();
104         busq2.doBuscar(this.mapa.getMapa(), texto);
105     },
106     doSyncGrid : function(grid, evtRI) {
107         console.log("sync grid");
108         selectionLayer = new esri.layers.GraphicsLayer();
109         selectionLayer.setRenderer(new esri.renderer.SimpleRenderer(new esri.symbol.SimpleFillSymbol().setColor(new dojo.Color([255, 255, 0, 0.50]))));
110         this.mapa.getMapa().addLayer(selectionLayer);
111         var rowId = grid.getItem(evtRI).OBJECTID;
112         selectionLayer.clear();
113         //add a new graphic to the selection layer
114         dojo.some(this.mapa.getMapa().graphics, function(graphic) {
115             if((graphic.attributes) && graphic.attributes.OBJECTID === evtRI) {
116                 var selectedState = new esri.Graphic(graphic.geometry).setAttributes(graphic.attributes);
117                 selectionLayer.add(selectedState);
118                 return true;
119             }
120         });
121     },
122     doGP : function(texto) {
123         console.log("tengo esto " + texto);
124         gp = new app.gp();
125         gp.doGP(this.mapa.getMapa(), texto);
126     }
127 };
128

```


identificar2.js

Tiene la responsabilidad de realizar la búsqueda y retornar los resultados según el punto señalado (*click*) por el usuario.

```
1  dojo.provide('app.identificar2');
2
3  dojo.declare("app.identificar2", null, {
4
5  doIdentificar : function(mapa, evt) {
6      query = new esri.tasks.Query();
7      query.geometry = evt.mapPoint;
8      templateBlq = new esri.dijit.PopupTemplate({
9          title : "Información",
10         fieldInfos : [{
11             label : "Número",
12             fieldName : "BLQ_PK_NUMERO",
13             visible : true,
14             format : {
15                 places : 0
16             }
17         }, {
18             label : "Nombre",
19             fieldName : "BLQ_NOMBRE",
20             visible : true,
21             format : {
22                 places : 0
23             }
24         }],
25         mediaInfos : [{
26             "title" : "",
27             "caption" : "",
28             "type" : "image",
29             "value" : {
30                 "sourceURL" : "http://www.eafit.edu.co/estudiantes/noticias/institucional/PublishingImages/biblioteca.jpg",
31                 "linkURL" : "http://www.eafit.edu.co/estudiantes/noticias/institucional/PublishingImages/biblioteca.jpg"
32             }
33         }, {
34             "title" : "",
35             "caption" : "",
36             "type" : "image",
37             "value" : {
38                 "sourceURL" : "http://ingenieria-matematica.eafit.edu.co/imagenes/menu/servicios.jpg",
39                 "linkURL" : "http://ingenieria-matematica.eafit.edu.co/imagenes/menu/servicios.jpg"
40             }
41         }
42     ]
43 });
44 templateEsp = new esri.dijit.PopupTemplate({
45     title : "Información",
46     fieldInfos : [{
47         label : "Espacio",
48         fieldName : "ESP_NOMBRE",
```

```

49         visible : true,
50         format : {
51             places : 0
52         }
53     },
54     },
55     mediaInfos : [{
56         "title" : "",
57         "caption" : "",
58         "type" : "image",
59         "value" : {
60             "sourceURL" : "http://ingenieria-matematica.eafit.edu.co/imagenes/menu/servicios.jpg",
61             "linkURL" : "http://ingenieria-matematica.eafit.edu.co/imagenes/menu/servicios.jpg"
62         }
63     }, {
64         "title" : "",
65         "caption" : "",
66         "type" : "image",
67         "value" : {
68             "sourceURL" : "http://www.eafit.edu.co/estudiantes/noticias/institucional/PublishingImages/biblioteca.jpg",
69             "linkURL" : "http://www.eafit.edu.co/estudiantes/noticias/institucional/PublishingImages/biblioteca.jpg"
70         }
71     }
72     ]
73 });
74 bloques = new esri.layers.FeatureLayer("http://b18p5sig8:8399/arcgis/rest/services/CampusSIGEAfit/mapaEAFITcampus_fullLayers/MapServer/184", {
75     mode : esri.layers.FeatureLayer.MODE_SELECTION,
76     infoTemplate : templateBlq,
77     outFields : ["*"]
78 });
79 espacios = new esri.layers.FeatureLayer("http://b18p5sig8:8399/arcgis/rest/services/CampusSIGEAfit/mapaEAFITcampus_fullLayers/MapServer/183", {
80     mode : esri.layers.FeatureLayer.MODE_SELECTION,
81     infoTemplate : templateEsp,
82     outFields : ["*"]
83 });
84
85 var def = [];
86 def.push(bloques.selectFeatures(query, esri.layers.FeatureLayer.SELECTION_NEW));
87 def.push(espacios.selectFeatures(query, esri.layers.FeatureLayer.SELECTION_NEW));
88
89 mapa.infoWindow.setFeatures(def);
90 mapa.infoWindow.show(evt.mapPoint);
91
92 }
93 });
94

```

busqueda2.js

Se encarga de realizar la búsqueda y retornar los resultados según los parámetros digitados por el usuario.

```
1 dojo.provide('app.busqueda2');
2
3 dojo.declare("app.busqueda2", null, {
4
5     doBuscar : function(mapa, txt) {
6
7         mapa.graphics.clear();
8         queryTask = new esri.tasks.QueryTask("http://b18p5sig8:8399/arcgis/rest/services/CampusSIGEAFFIT/mapaEAFITcampus_fullLayers/MapServer/184");
9         query = new esri.tasks.Query();
10        query.text = txt;
11        query.returnGeometry = true;
12        query.outFields = ["OBJECTID", "BLQ_NOMBRE", "BLQ_NOMBRE2"];
13        //query.where = "BLQ_NOMBRE =" + txt + "'";
14        query.where = "BLQ_NOMBRE LIKE '%" + txt + "%'";
15        queryTask.execute(query, function(results) {
16
17            /* for (var i=0, il=results.features.length; i<il; i++) {
18                var featureAttributes = results.features[i].attributes;
19                for (att in featureAttributes) {
20                    console.log(" " + att + " " + featureAttributes[att] + "");
21                }
22            } */
23            //build an array of attributes
24            var items = dojo.map(results.features, function(feature) {
25                console.log(feature.attributes);
26                return feature.attributes;
27            });
28            var data = {
29                identifier : "OBJECTID",
30                items : items
31            };
32            store = new dojo.data.ItemFileReadStore({
33                data : data
34            });
35            grid.setStore(store);
36            grid.setSortIndex(1, "true");
37            //sort on the state name
38            dojo.forEach(results.features, function(feature) {
39                mapa.graphics.add(feature);
40            });
41        });
42    }
43 });
```

ANEXO B. CONCEPTOS SOBRE LENGUAJE JAVASCRIPT

En este anexo se recogen conceptos de este lenguaje que son importantes para el desarrollo de una aplicación del lado del cliente (*front-end*) y que lo diferencian de otros. Para ampliar la información puede remitirse al texto de Stefanov, (2008).

JavaScript es un lenguaje que nació de la mano de los navegadores Web, en los años 90s, con el propósito de ofrecerle al usuario una interacción más rica y evitar tareas del lado del servidor, tales como validaciones en formularios. Su uso se extendió, sin embargo enfrentó dificultades debidas a que los fabricantes carecían de estándares, ocasionando incompatibilidades, por lo que si un código funcionaba en un *browser* en otro probablemente no. Otra de las razones por las cuales adquirió mala reputación fue por su abuso en páginas para adicionar animaciones y otros efectos, afectando seriamente la usabilidad.

A pesar del panorama, JavaScript fue evolucionando, en gran parte debido a la aparición, maduración y estandarización de los navegadores, convirtiéndose en un potente lenguaje Orientado a Objetos. De esta manera la comunidad desarrolladora encontró una nueva y mejor forma de programar aplicaciones ricas, del lado del cliente.

Funciones son datos

Esto significa que las dos siguientes formas de definir una función son exactamente iguales:

```
function f(){return 1; }
```

```
var f = function {return 1; }
```

La segunda forma es denominada notación literal.

Funciones anónimas

Las funciones pueden utilizarse sin asignarles un nombre:

```
function (a){return a; }
```

Se emplean básicamente en dos casos:

- Para pasarla como parámetro a otra función y que ésta la ejecute
- Se define y se ejecuta en seguida.

Funciones tipo Callback

Cuando se pasa una función A, a otra función B y B la ejecuta. A es llamada función *callback*. Son útiles porque:

- Permiten pasar funciones sin necesidad de nombrarlas (lo que disminuye el número de variables globales).

- Se delega la responsabilidad de llamar una función a otra función (hay menos código).
- Pueden ayudar con el *performance*.

Funciones que se auto-invocan

En el siguiente ejemplo, se define una función anónima dentro de paréntesis, seguido por otro par de paréntesis, que hace que se ejecute inmediatamente y es el lugar para los parámetros que la función acepta.

```
(
  function (name){
    alert('Hello ' + name + '!');
  }
)('dude')
```

Este tipo de funciones son adecuadas para tareas de inicialización o que realizan por una sola vez.

Funciones privadas (inner)

Se da cuando se tiene una función global *a()* e internamente se define una función *b()*, ésta es de tipo local y no es accesible fuera de *a()*. *b()* es denominada función privada.

```
function a (param){
  function b (theinput){
    return theinput*2;
  };
  return 'The result is ' + b(param);
};
```

Son dos beneficios que se obtiene al utilizar funciones privadas:

- Mantener el espacio global de nombres limpio (se disminuyen las posibilidades de colisiones).
- Se exponen solo ciertas funciones al exterior.

Scope - Alcance de variable

En el lenguaje, se refiere a *scope function* cuando una variable es definida dentro una función y ésta no es visible fuera de ella. Sin embargo, si se declaran en bloques de sentencias *if* o *for*, si son visibles para el resto.

Las *variables globales*, se definen fuera de cualquier función y las *variables locales* deben estar contenidas en una. El código de una función tiene acceso tanto a variables globales como locales.

Es de anotar que si se declara una variable y no se utiliza la sintaxis *var*, es asignada automáticamente a alcance global.

En este aspecto se debe tener en cuenta los siguientes puntos:

- Minimizar el número de variables globales
- Siempre declarar las variables con *var*

En fragmento de código a continuación se muestra un posible conflicto entre el alcance global y el local:

```
var a = 123;

function f(){

    alert (a);

    var a = 1;

    alert (a);

}

f();
```

Se espera que el resultado del primer *alert* sea 123 y el del segundo 1. Pero no es el caso, el primero será "undefined", ya que dentro de una función el alcance local es más importante que el global. Entonces la local sobrescribe la global que tenga el mismo nombre. Por esto, en el primer aviso, *a* no estaba aún definida, pero ya existía en el espacio local.

Closures

Dos conceptos a destacar antes de estudiar las *closures*:

- *Scope chain*: si se define una función *n()* dentro de una *f()*, *n()* tiene acceso a las variables de su propio *scope*, más el de sus "padres". Ejemplo

```
var a = 1;

function f(){

    var b = 1;

    function n(){

        var c = 3;
```

```
    }  
  }
```

- *Lexical scope*: las funciones crean su ambiente (*scope*) cuando se definen, no cuando se ejecutan. Ejemplo:

```
function f1(){ var a = 1; f2(); }  
function f2(){ return a; }  
f1();
```

El resultado es:

a is not defined

Dentro de `f1()` se llama a `f2()`. Dado que la variable local está dentro de `f1()` se podría esperar que `f2()` tuviera acceso, pero no se cumple en este caso. Cuando `f2()` fue *definida* no está `a` a la vista. `f2()`, tal como `f1()`, solo tiene acceso a su propio *scope* y al global. `f1()` y `f2()` no comparten sus *scopes* locales.

Cuando una función se define, “recuerda” su ambiente, es decir su *scope chain*. Pero esto no significa que la función también recuerde cada una de las variables en su *scope*. Al contrario, se pueden adicionar, remover o actualizar variables dentro del *scope* de la función- y ésta tomará el último estado de ellas. Si en el ejemplo anterior se declara una variable global `a`, `f2()` la puede ver, ya que conoce el camino y tiene acceso a todo el ambiente global. También se debe notar que `f1()` incluye un llamado a `f2()`, y funcionará aunque `f2()` no esté definida aún. Todo lo que `f1()` necesita es conocer su *scope*, y lo que esté dentro, lo tendrá a su disposición. Ejemplo:

```
function f1(){ var a = 1; return f2(); }  
function f2(){ return a; }  
f1();
```

El resultado es:

a is not defined

```
var a = 5;  
f1();
```

El resultado es:

5

```
a = 55;
```

```
f1();
```

El resultado es:

55

```
delete a;
```

```
true
```

```
f1();
```

El resultado es:

a is not defined

Este comportamiento le da a JavaScript gran flexibilidad, se pueden adicionar y remover variables. Por ejemplo:

```
delete f2();
```

```
true
```

```
f1();
```

f2 is not defined

```
var f2 = function () { return a * 2;}
```

```
var a = 5;
```

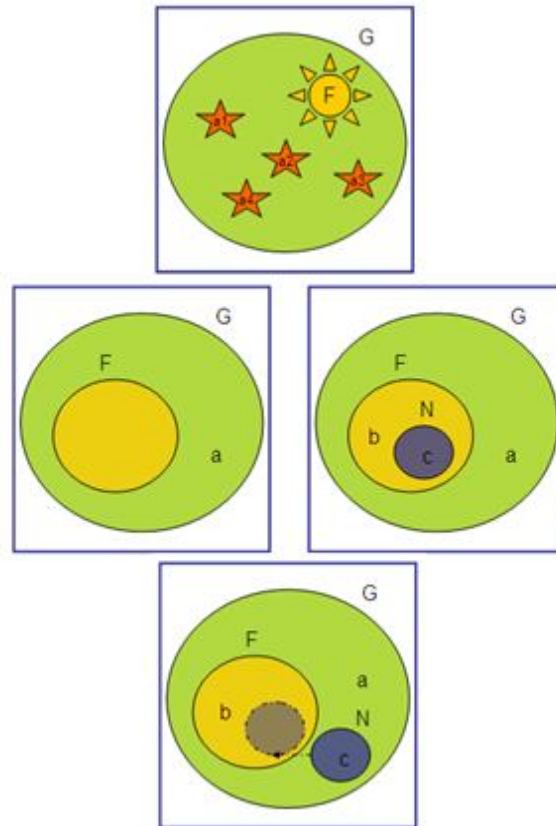
```
f1();
```

10

f1() funciona, porque lo único que necesita conocer es cómo tener acceso a su *scope* y no lo que su *scope* contenía en un punto determinado.

Utilización de *Closures*

Los siguientes gráficos introducen el concepto de *closures*, donde el *scope* global se asemeja al universo que lo contiene todo, incluidas variables como *a* y funciones como *F*. Las funciones tienen su propio espacio privado y pueden usarlo para almacenar otras variables (y funciones).



Concepto de closures. Tomado de (Stefanov, 2008)

De acuerdo a la figura BNC, en el tercer cuadro, en el punto a , se está dentro del espacio es global. En b , se tiene acceso al global y al de la función F . En c (que está dentro de N), el acceso puede ser global, al espacio de F y al de N . No se puede ir de a hacia b porque b es invisible fuera de F . Pero si, de c a b o de N a b .

El último cuadro, es el *closure*, ocurre cuando N sale de F y termina en el espacio global. Lo que pasa es que N está en el mismo espacio global de a y dado que las funciones recuerdan el ambiente en el cual fueron definidas, N aún tendrá acceso al espacio de F , y desde aquí a b .

Los mecanismos por los cuales N hace el *closure* es haciéndose a sí misma global (omitiendo *var*) o haciendo que F la entregue (*return*) al espacio global. Los siguientes ejemplos ponen esto en práctica:

Closure # 1

```
function f(){
    var b = "b";
    return function (){
        return b;
    }
}
```

```
    }  
}
```

Esta función contiene una variable *b*, que es local, por tanto es inaccesible desde el espacio global.

El valor de retorno de *f()* es otra función, que tiene acceso al espacio privado y al global, entonces puede ver a *b*. Al ser *f()* una función global puede ser llamada y asignarle el valor retornado a otra variable global:

```
var n = f();
```

```
n();
```

El resultado es:

“b”

Closure # 2

El resultado es el mismo del closure # 1, pero en lugar de retornar una función, se crea una función global *n()* dentro del cuerpo.

```
var n;
```

```
function f(){  
    var b = “b”;  
    n = function (){  
        return b;  
    }  
}
```

```
}
```

Al invocar *f()*, la nueva función definida dentro de ella al no tener la palabra reservada *var*, se toma como global. En tiempo de definición, al estar *n()* contenida dentro de *f()*, tiene acceso a su *scope*, aunque también sea parte del espacio global.

El resultado de *n()* es:

“b”

Closure # 3

Cuando se pasa un argumento a una función, se pone a disposición como variable local. Se puede crear una función que retorne otra función, que retorne el argumento de sus padres.

```
function f(arg){
```

```

    var n = function (){
        return arg;
    };
    arg++;
    return n;
}

```

Al utilizar la función:

```

var m = f(123);
m();

```

El resultado es:

“124”

Se nota que *arg* se incrementó después de que la función fue definida y cuando es llamada, *m()* retorna el valor actualizado. Esto demuestra la manera en que la función se une a su *scope* y a los valores que se encuentran en él.

Closure en un Loop

Este es un caso que dificulta darse cuenta de errores, ya que a simple vista todo parece normal.

Hay un bucle, que se ejecuta 3 veces, en cada una se crea una nueva función que retorna el número de la secuencia del ciclo. La nueva función se adiciona a un arreglo que será retornado al final.

```

function f(){
    var a = [];
    var i;
    for (i=0; i<3; i++){
        a[i] = function(){
            return i;
        }
    }
    return a;
}

```

```
}
```

Se corre la función y se asigna el resultado al arreglo *a*:

```
var a = f();
```

Entonces, se tiene un arreglo de tres funciones, el resultado de recorrerlo elemento por elemento es:

```
a[0]()
```

```
3
```

```
a[1]()
```

```
3
```

```
a[2]()
```

```
3
```

Este no era el comportamiento esperado, lo que sucede es que se crearon tres *closures* que apuntan a la misma variable local *i*. *Closures* no recuerdan el valor, sólo referencian la variable *i* y retornan su valor actual. Después del loop, el valor de *i* es **3**, de modo que las tres funciones apuntan al mismo valor.

Entonces, para implementar el comportamiento correcto, de una forma elegante, se necesitan tres variables diferentes y usar otro *closure*:

```
function f(){  
    var a = [];  
    var i;  
    for (i=0; i<3; i++){  
        a[i] = (function(x){  
            return function(){  
                return x;  
            }  
        })(i);  
    }  
    return a;  
}
```

Al probarla, los resultados serán:

```
var a = f();
```

```
a[0]()
```

```
0
```

```
a[1]()
```

```
1
```

```
a[2]()
```

```
2
```

En este caso, en lugar de crear solo una función que retorne *i*, se pasa a otra función auto-ejecutable, para la cual *i* se convierte al valor local *x*, que tiene un valor diferente en cada iteración.

Otra alternativa es utilizar una función interna “normal”, que localizará el valor de *i* cada iteración:

```
function f(){  
    function makeClosure(x){  
        return function(){  
            return x;  
        }  
    }  
    var a = [];  
    var i;  
    for (i=0; i<3; i++){  
        a[i] = makeClosure(i);  
    }  
    return a;  
}
```

Getter/Setter

Se aplica cuando se tiene una variable que contendrá un rango específico de valores y no se debe exponer a que en alguna parte del código se altere. Se protege la variable dentro de una función y se proporcionan dos funciones adicionales, *get* y *set*:

```

var getValue, setValue;;
(function (){
    var secret = 0;
    getValue=function(){
        return secret;
    }
    setValue=function(v){
        secret=v;
    };
})();

```

En este caso, la función que contiene todo es una función anónima auto-invocada. Se definen setValue() y get Value como funciones globales, mientras la variable secret se mantiene local e inaccesible directamente.

Objetos en JavaScript

Los objetos se asemejan a los arreglos, se diferencian en que se especifican las claves, y contienen propiedades que pueden ser funciones. Ejemplo de declaración:

```

var hero = {
    breed: 'Turtle',
    occupation: 'Ninja',
    say: function(){
        return 'I am' + hero.occupation;
    }
}

```

Al invocar *hero.say()* el resultado es "I am Ninja "

Cabe anotar que se utiliza la notación de puntos (.) para acceder a los métodos y propiedades.

La siguiente pieza de código emplea el valor *this*, que permite tener acceso al objeto, aunque se esté adentro de una función. Cuando se utiliza *this*, se hace referencia al "objeto actual".

```

var hero = {

```

```

    name: 'Rafaelo',
    sayName: function(){
        return this.name;
    }
}

```

Al invocar *hero.sayName()* el resultado es "Rafaelo".

Prototype

En JavaScript todas las funciones tienen una propiedad llamada *prototype*, que inicialmente contiene un objeto vacío, al cual se le pueden adicionar métodos y propiedades, incluso se puede reemplazar completamente por otro objeto. Ejemplo:

```

function Gadget (name, color){
    this.name =name;
    this.color=color;
    this.whatAreYou = function (){
        return 'I am a ' + this.color + ' ' + this.name;
    }
}

Gadget.prototype.price=100;
Gadget.prototype.rating=3;
Gadget.prototype.getInfo=function(){
    Return 'rating:' + this. rating+ ', price:' + this.price;
};

```

Si se crea un objeto *newtoy* utilizando el constructor *Gadget*, se puede acceder a todos los métodos y propiedades definidas anteriormente:

```

var newtoy = new Gadget('webcam', 'black');
newtoy.price;

```

El resultado es **100**

Cabe anotar que en JavaScript los objetos son pasados por referencia, entonces el *prototype* no es copiado en cada nueva instancia. Esto significa que se puede

modificar el *prototype* en cualquier momento y todos los objetos (incluso aquellos creados antes), heredarán los cambios.

XMLHttpRequest

XMLHttpRequest es un objeto que permite enviar peticiones http desde JavaScript, su forma abreviada de notarlo es XHR. Su nacimiento se dio en las aplicaciones AJAX, en las cuales ante la respuesta a una petición, se refresca solo una parte de la página. Ejemplo:

Envío de la petición

```
var xhr = new XMLHttpRequest();  
  
xhr.onreadystatechange = myCallback; // Se adiciona el listener  
  
xhr.open('GET', 'somefile.txt', true);  
  
xhr.send('');
```

Procesado de la respuesta

El listener *onreadystatechange*, está vinculado a la propiedad *readyState* del objeto XHR, sus posibles valores son: 0 sin inicializar, 1 cargando, 2 cargado, 3 interactivo y 4 completo. El valor 4 significa la llegada de la respuesta y está lista para ser procesada.

En la función *myCallback* se debe verificar el código de estatus de la petición http, como por ejemplo 404 (archivo no encontrado) o 200 (OK).

Recepción de la respuesta:

```
function myCallback (){  
    if(xhr.readyState < 4){  
        return; // no está listo aún  
    }  
  
    if(xhr.status !== 200){  
        alert('Error!'); // http estatus no está OK  
        return;  
    }  
  
    // funcionó  
    alert(xhr.responseText);  
}
```


BIBLIOGRAFIA

Acevedo, C., Guzmán, I. (2006). "*Modelo de Referencia para el desarrollo de servicios y aplicaciones para dispositivos Móviles*". Tesis de pregrado. Facultad de Ingeniería Electrónica y Telecomunicaciones. Universidad del Cauca.

Anta, J.A.(2012). "*Desarrollo de Aplicaciones Móviles que Merezcan la Pena*". ESRI España. Disponible en: <http://www.slideshare.net/ESRI/seminario-desarrollo-de-aplicaciones-mviles-que-merezcan-la-pena-valencia-2012>

Bartley, J., Hutchins, K., Ponnusamy, P. (2010). "*Patterns and Best Practices when using the ArcGIS API for JavaScript*". ESRI Developer summit. Disponible en:http://proceedings.esri.com/library/userconf/devsummit10/papers/tech/patterns_and_best_practices_for_building_applications_with_the_arcgis_api_for_javascript.pdf

Buckley, D. (1997). "*The GIS Primer. An introduction to Geographic Information Systems*". Disponible en: http://bgis.sanbi.org/GIS-primer/page_01.htm

Calero, A., Abadía, J.(2012). "*ArcGIS for Server*". Conferencia 12 ESRI España. Disponible en: <http://www.slideshare.net/ESRI/ags-ce12-fina>

Carrillo, G. (2012). "*Web mapping client comparison v.6*". Disponible en: <http://geotux.tuxfamily.org/index.php/en/geo-blogs/item/291-comparacion-clientes-web-v6>

Chandan, R. (2011). "*Sencha / ExtJS : Object Oriented JavaScript*". Disponible en: <http://www.slideshare.net/rohan.chandane/sencha-extjs-object-oriented-javascript>

Chivite, I., Zwaap, R. (2011). "*Creating Web Mapping Applications with ArcGIS*". ESRI Federal User Conference. Disponible en: http://proceedings.esri.com/library/userconf/feduc11/papers/tech/2011-feduc_creating-web-mapping-applications-with-arcgis.pdf

Davis, S. (2007). "*GIS for Web Developers: Adding 'Where' to Your Web Applications*". Pragmatic Bookshelf

Dragicevic, S. (2004). "*The potential of Web-based GIS*". Springer Berlin / Heidelberg, 6(Number 2 / June, 2004), 79-81.

ESRI - ArcGIS <http://www.esri.com/software/arcgis/arcgisserver>

Fernández López, G., Méndez Pérez, X. (2011). "*Curso de introducción a OpenLayers*". Xeoinquedós. Disponible en: <http://blog.sonxurxo.com/wp-content/uploads/2011/05/presentacion.pdf>

Forbes, B. (2011). "*Modular JavaScript AMD & CommonJS modules*". Disponible en: <http://bryanforbes.github.com/amd-commonjs-modules-presentation/2011-10-29/#0>

Foster, S. (2011). "*Application Controller*". Dojo Foundation. Disponible en: http://dojotoolkit.org/documentation/tutorials/1.6/recipes/app_controller/

Free and Open Source for Geospatial Conference <http://foss4g.org/>

Fu, P., Sun, J. (2011). "*Web GIS: Principles and Applications*". ESRI Press

Gaona, M. (2011). "*Metodologías de Desarrollo de Software*". Tendencias en ingeniería de software. Escuela de Ingeniería de Sistemas y Computación Facultad de Ingeniería. Universidad del Valle. Disponible en: <http://eisc.univalle.edu.co/cursos/web/material/750280/1/Metodologias-de-Desarrollo-de-Software.pdf>

Garbin, D.A., Fisher, J.L. (2010). "*Open source for enterprise Geographic Information Systems*". IT Pro Noviembre/Diciembre. IEEE Computer Society.

GeoExt <http://geoext.org/>

Geomajas <http://www.geomajas.org>

GeoServer <http://geoserver.org>

GISLAN Geographic Applications SL <http://www.gislan.com/es>

González, L., Rienzi, B., Sosa, R. (2011). "*Conceptos Introductorios*". Taller de Sistemas de Información Geográficos Empresariales. Facultad de Ingeniería - Universidad de la República. Uruguay Disponible en: <http://www.fing.edu.uy/inco/cursos/tsi/TSIG/clases2011/ConceptosIntroductorios2011.pdf>

Goodchild, M.F. (1997). "*What is geographic information science?*". Disponible en: <http://www.ncgia.ucsb.edu/giscc/units/u002/u002.html>

Graser, A. (2010). "Geoserver vs. Mapserver". Disponible en: <http://underdark.wordpress.com/2010/06/08/geoserver-vs-mapserver/>

Harmes, R., Diaz, D. (2007). "*Pro JavaScript Design Patterns*". Apress

Heilmann, C. (2009). "*Javascript Best Practices*". Disponible en: <http://www.slideshare.net/cheilmann/javascript-best-practices-1041724>

Infraestructura Colombiana de Datos Espaciales <http://www.icde.org.co/web/guest/inicio>

Infraestructura de Datos Espaciales de España <http://www.ideo.es/>

Jornadas SIG Libre - SIGTE Girona <http://www.sigte.udg.edu/jornadassiglibre/>

Korte, G. 2001. "*The GIS Book*". (5th Ed. Rev.). Autodesk Press.

Lamas, A., García, M. (2011). "*Cliente de ikiMap para Android utilizando librería SIG Mini*". 7as Jornadas Internacionales gvSIG. Disponible en http://downloads.gvsig.org/download/events/gvSIG-Conference/7th-gvSIG-Conference/reports/7j-Cliente_ikiMap.pdf

Longley, P., Goodchild, M., Maguire, D., Rhind, D. (2010). "*Geographic Information Systems and Science*". 3rd Edition. Wiley

Mapbender <http://www.mapbender.org/>

Mapfish <http://mapfish.org/>

MapServer open source web mapping <http://mapserver.org>

Mitchell, T. (2005). "*Web Mapping Illustrated*". O'Reilly

Murphey, R. (2010). "*Dojo Confessions*". Disponible en: <http://www.slideshare.net/rmurphey/dojo-confessions>

NCGIA. National Center for Geographic Information and Analysis. <http://www.ncgia.ucsb.edu/>

Nebert, D. (ed.) (2004). "*Developing Spatial Data Infrastructures: The SDI Cookbook. Version 2.0. [S.I.]: Global Spatial Data Infrastructure*". Disponible en: <http://www.gsdi.org/docs2004/Cookbook/cookbookV2.0.pdf>

Neteler, M., Mitasova, H. (2008). "*Open Source GIS: A GRASS GIS Approach*". Third Edition The International Series in Engineering and Computer Science: Volume 773. Springer, New York

Nivala, A.M., Brewster, S., Sarjakoski, T. (2008). "*Usability Evaluation of Web Mapping Sites*". The British Cartographic Society. The Cartographic Journal Vol. 45 No. 2 pp. 129–138

OGC (Consortio para los Sistemas Geoespaciales abiertos) <http://www.opengeospatial.org/>

Olaya, V. (2011). "*Sistemas de Información Geográfica*". Disponible en: http://wiki.osgeo.org/wiki/Libro_SIG

OpenLayers: Free Maps for the Web <http://openlayers.org/>

OpenGeo <http://opengeo.org/>

Oracle <http://www.oracle.com/index.html>

Oracle JSFConvert. (2009). "JAVA Shapefile Converter". Disponible en: <http://www.oracle.com/technetwork/database/enterprise-edition/downloads/jsfconvert-readme-129638.pdf>

Osmani, A. (2012). "Learning JavaScript Design Patterns". Volume 1.5.2. Disponible en: <http://addyosmani.com/resources/essentialjsdesignpatterns/book/>

Peng, Z.R., Tsou, M.H., (2003). "Internet GIS: distributed geographic information services for the Internet and wireless networks". John Wiley & Sons Inc.

Pereyra, A. (2008). "Metodología para el proceso de creación de Aplicaciones Web". Disponible en: <http://www.slideshare.net/Yaraher/metodologia-para-creacion-de-aplicaciones-web>

PostGIS [http:// postgis.refrains.net/](http://postgis.refrains.net/)

Rational (1998), "Rational Unified Process Best Practices for Software Development Teams". Rational Software White Paper. Disponible en: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Runeson, P., Greberg, P. (). "Extreme Programming and Rational Unified Process – Contrasts or Synonyms? ". Lund University. Sweden. Disponible en: http://acishost.acis.org.co/fileadmin/Curso_Memorias/Curso_CMMI_Sep06/Modulo%20%20-%20Product%20Engineering%20/xp_rup.pdf

Rusell, M.(2008). "Dojo The definitive Guide". O'Reilly.

Sanz, J., Montesinos, M., (2009). "Panorama actual del ecosistema del SIG libre". Revista Novática 198.

Schwaber, K. (1997). "SCRUM Development Process". Advanced Development Methods. Disponible en: <http://gowegian.5gbfree.com/scrumDevelopmentProcess.pdf>

Shen, S., Cheng, X., Gong, P., (2008). "Sensor Web Oriented Web-Based GIS". In Proceedings of the 8th International Symposium on Web and Wireless Geographical Information Systems. Shanghai, China: Springer-Verlag, pp. 86-95. Disponible en: <http://portal.acm.org/citation.cfm?id=1504895>

SIGTE - Servei de Sistemes d'Informació Geogràfica i Teledetecció - Universitat de Girona http://www.sigte.udg.edu/sigte_es/

SitePen <http://www.sitepen.com/about/index.html>

Stefanov, S. (2008). "Object-Oriented JavaScript Create scalable, reusable high-quality JavaScript applications and libraries". Pack Publishing.

The Open Source Geospatial Foundation <http://www.osgeo.org/>

Thomas, J. (2011). "Moving to Dojo 1.7 and the path to 2.0". Disponible en: <http://www.slideshare.net/jthomas/moving-to-dojo-17-and-the-path-to-20>

TIOBE Software. The Coding standards company
<http://www.tiobe.com/index.php/content/company/Home.html>

Valencia A., A.M. (2010). "Estrategia de integración de una base de datos georreferenciada con los sistemas corporativos". Proyecto de grado Ingeniería de Sistemas. Universidad EAFIT.

Valencia M. de A., J. (2008). "Pasado, presente y futuro de las Infraestructuras de datos espaciales". Disponible en: <http://www.bubok.es/libros/210512/PASADO-PRESENTE-Y-FUTURO-DE-LAS-INFRAESTRUCTURAS-DE-DATOS-ESPACIALES>

Velarde, V. (2011). "Desarrollo de aplicaciones geográficas web: OpenLayers". Curso de experto en desarrollo y gestión de Sistemas de información geográfica. Universidad de Cantabria. Disponible en: <http://victorvelarde.files.wordpress.com/2011/01/curso-openlayers-victorvelarde.pdf>

You, M., Chun-wen, C., Hantsai, L., Hsuan, L. (2007). "A usability evaluation of web map zoom and pan functions". International Journal of Design, 1(1), 15.