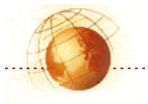


Capítulo 2
Marco Teórico

[2. Marco Teórico]

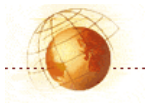
En este capítulo se tratan y definen los conceptos y tecnologías que establecen el contexto de trabajo de esta tesis. Específicamente se tratan tecnologías SIG, tecnologías de ingeniería de *software*, y tecnologías *web*. Todas estas tecnologías son importantes para este proyecto de tesis porque son las bases que permitieron su desarrollo, así como para muchos otros proyectos como se verá en el capítulo 3.

2.1 Tecnologías SIG

Esta parte del capítulo se enfoca a las tecnologías SIG que conforman el contexto de trabajo de este proyecto de tesis. Se divide en 3 secciones principales que son: los Sistemas de Información Geográfica, *Geography Markup Language*, y *Features* o rasgos geográficos.

2.1.1 Sistemas de Información Geográfica

En la actualidad son muchos los sistemas de información que nos rodean. La principal función de ellos es el proveernos de herramientas que nos den soporte para la toma de decisiones. Particularmente, los Sistemas de Información Geográfica (SIG) son un tipo especializado de sistema que se distinguen por su capacidad de manejar información espacialmente referenciada, es decir, ligada a una localización específica de la tierra a través de sistemas de coordenadas, y que además permiten su visualización.



Decimos que son herramientas porque ayudan a la formación de elementos de juicio para la toma de decisiones, después de que se han aprovechado sus funciones de captura, almacenamiento, refinamiento, análisis y visualización de la información.

Los SIG integran operaciones comunes de bases de datos tales como consultas y análisis estadísticos, con la visualización y los beneficios geográficos que son ofrecidos por los mapas. Estas características son las que distinguen a los SIG de los demás sistemas de información las cuales amplían su uso en empresas públicas y privadas para explicar sucesos, predecir resultados y planificar estrategias relacionados con un dominio específico.

Los SIG pueden organizar y procesar datos provenientes de una gran cantidad de fuentes para su explotación. Por ejemplo, pueden procesar datos obtenidos de mapas, imágenes y fotografías, datos estadísticos generados a partir de análisis matemáticos, datos provenientes de sistemas *CAD* (*computer-assisted design*) y de bases de datos [Pech, 2002]. Si se desea mayor información acerca de los GIS se puede consultar [GIS, 2004].

2.1.1.1 Concepto SIG

La diversidad de los SIG ha propiciado la proliferación de una gran variedad de definiciones. Generalmente un usuario define un SIG de acuerdo al uso que éste le dé y a su propia experiencia y habilidades.

El instituto de investigación de sistemas ambientales (ESRI) define a un SIG como “una colección organizada de *hardware, software,* datos geográficos y de personal diseñada para capturar, almacenar, actualizar, manipular, analizar y mostrar eficientemente todas las formas de información referenciada geográficamente” [ESRI, 2004].

Así mismo, el sistema corporativo de información geográfica de PEMEX (SICORI) define a un SIG como “una disciplina basada en conocimientos, metodologías y procedimientos asistidos por computadora, que permiten la incorporación, almacenamiento, manipulación, procesamiento, consulta y presentación de

información referenciada geográficamente en formatos gráficos y no gráficos" [SICORI, 2001].

En el Laboratorio de Tecnologías de Geoinformación (GISUDLA), lo definimos como un conjunto de herramientas, que haciendo uso de diversas tecnologías, nos provee mecanismos de adquisición, almacenamiento, interpretación, **distribución**, procesamiento, consulta y visualización de información referenciada espacialmente, para proporcionarnos las bases de un juicio en la toma de decisiones.

2.1.1.2 Componentes

Para comprender mejor el funcionamiento de un SIG es importante conocer cuales son los componentes que lo constituyen. Un SIG se puede ver como una caja negra, pues manipula datos reales, en nuestro caso localizaciones de entidades en un espacio físico.

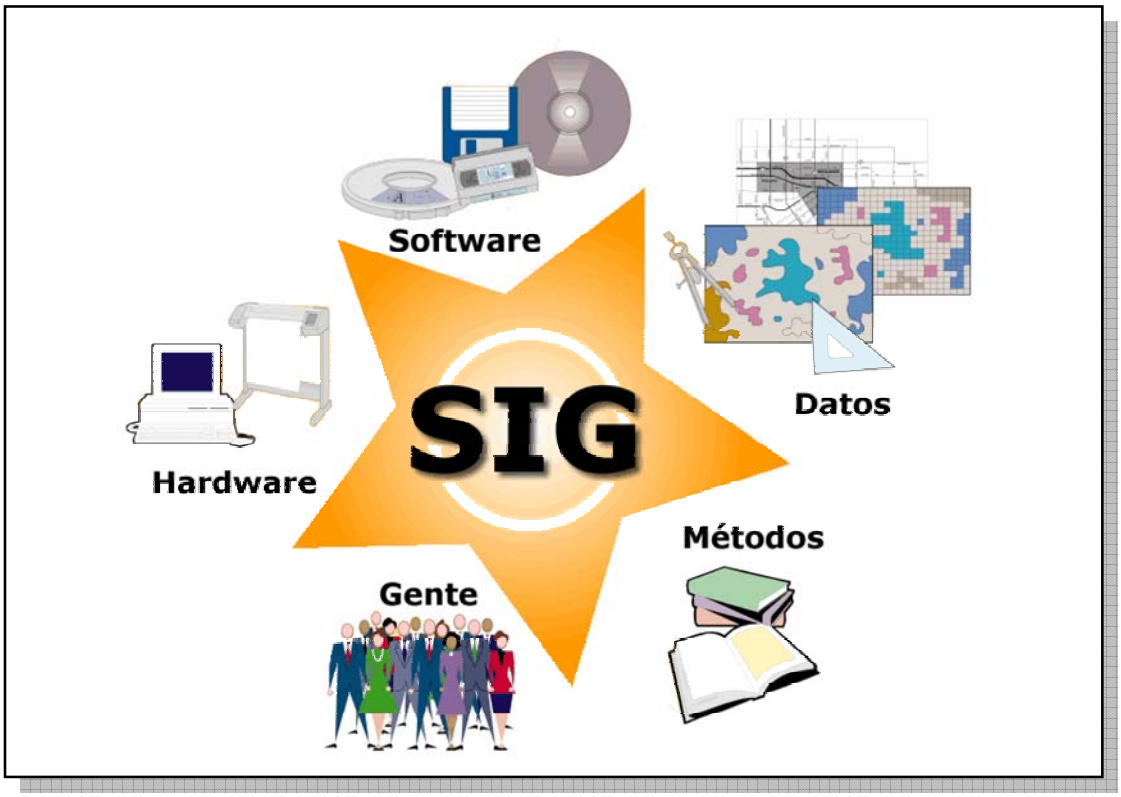


Figura 2.1 – Componentes GIS

En la figura 2.1 podemos ver los componentes que conforman a un SIG, y a continuación se explica cada uno de estos componentes.

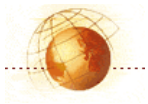
- **Hardware** – Es el equipo y los periféricos en los cuales el SIG opera, puede ser desde una estación de trabajo hasta un servidor centralizado. Los equipos pueden operar en modo *stand-alone* o *networked-configuration*. Algunos ejemplos de hardware pueden ser: computadoras, redes, dispositivos periféricos, impresoras, plotters y digitalizadores.

- **Software** – Es el que provee las funciones y herramientas que el usuario necesita para almacenar, analizar, y mostrar la información geográfica. Los componentes de software clave son: Software SIG, Software de BD, Software de SO y Software de red, recordando que el Software SIG se considera al Software comercial y al Software abierto.

- **Datos** – Uno de los componentes más importantes de un SIG son los datos. Es absolutamente esencial que la información sea precisa. Algunos diferentes tipos de datos son: Datos vectoriales, Datos raster, Datos de imágenes, Datos de atributos.

- **Gente** – La tecnología SIG es claramente de valor limitado sin la gente que administre el sistema y desarrolle planes para su aplicación. Los usuarios SIG pueden ser desde especialistas técnicos calificados hasta planeadores y analistas de mercado quienes usen los SIG para ayudarse con sus tareas diarias. Ejemplo de gente SIG pueden ser: Administradores, Gerentes, Técnicos SIG, Expertos de aplicación, Usuarios finales, Consumidores, Geólogos, Urbanistas.

- **Métodos** – Los métodos son planes bien diseñados y reglas de negocio específicas de la aplicación que describen como la tecnología es aplicada. Esto incluye: Pautas, Especificaciones, Estándares y Procedimientos.



2.1.2 *Geography Markup Language* (GML)

Como ya sabemos, XML se ha convertido en la base común de muchas aplicaciones. Como resultado muchas aplicaciones están disponibles para trabajar con archivos XML y por consiguiente con GML (una extensión de XML). El objetivo de esto es que usando un estándar de la herramienta XML es posible hacer muchas de las mismas cosas que costaban tanto trabajo hace un par de años.

GML es una especificación para la codificación en XML (eXtensible Markup Language) [XML, 2004] para el transporte y almacenamiento de información geográfica, incluyendo las propiedades espaciales y no espaciales de los llamados *features* o rasgos geográficos [GML, 2002]. GML está siendo desarrollado por el OpenGIS Consortium (OGC) y está respaldado por un gran número de empresas dentro de las que están Oracle Corporation, Galdos Systems Inc., MapInfo, cubeWerx, sólo por mencionar algunas.

[Prins, 2003] define que GML es "un medio estandarizado de almacenamiento de información geográfica en archivos codificados en XML especificados por el OpenGIS Consortium". XML, es un formato abierto basado en ASCII, que usa etiquetas descriptivas para almacenar los datos donde las etiquetas pueden estar anidadas dentro de otras. En la figura 2.2 muestra el ejemplo de un GML muy sencillo, es la descripción de la ciudad de Puebla asociado a un lugar de la tierra cuyas coordenadas latitud, longitud son {-98.19294739,19.04863167}, pertenece al sistema de referencia espacial 32013 y proporciona información acerca de este punto como nombre, estado, coordenadas de la ciudad y población aproximada.

Con respecto a la visualización, ésta no requiere del todo una herramienta GIS, porque una simple transformación de GML a otro formato basado en XML como *Scalable Vector Graphics* (SVG) [SVG, 2001] o *eXtensible 3D* (X3D) [X3D, 2003] puede hacerse a través de *scripts*, de archivos *eXtensible Stylesheet Language* (XSL) [XSL, 2003] describiendo como representar los *features* de GML y un motor de transformación común disponible con el que cuentan las plataformas construidas hoy en día. Todo esto es posible porque GML ha estandarizado la manera en que se describen los *features* geográficos. Inclusive, los navegadores web pueden usar esas transformaciones. Es importante señalar que ya hay módulos de visualización

bastante desarrollados que se pueden aprovechar, algunos se explican en el capítulo 3.

```
<gml:featureMember>
  <gml:Point srsName=" 32013">
     <gml:coordinates decimal="." cs="," ts="">
       -98.19294739,19.04863167
     </gml:coordinates>
  </gml:Point>
  <Nombre>Puebla de Zaragoza</Nombre>
  <Estado>Puebla</Estado>
  <Poblacion>25698744</Poblacion>
</gml:featureMember>
```

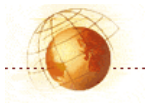
Figura 2.2 – Ejemplo de GML

Además, GML así como cualquier otra codificación XML, representa información geográfica en la forma de texto. Esto agrega simplicidad y visibilidad de su lado, pues es muy fácil de examinar y manipular.

Las definiciones GML se describen usando esquemas XML estándares. Los esquemas XML especifican las reglas y formatos a usar para codificar *features* espaciales abstractos, como son líneas y polígonos en GML (en la siguiente sección veremos que es un *feature* geográfico). Se pueden definir más esquemas de aplicación específicos o documentos de definición de tipo (DTD) que se ajusten a nuestro modelo de datos extendiendo los esquemas GML disponibles [Prins, 2003].

Aparte de describir al mundo real, los *features* y los atributos GML pueden también ser usados para describir operaciones espaciales y consultas de atributos y especificar argumentos, comúnmente *bounding boxes* y otras geometrías para estas operaciones. Actualmente GML es el más usado como formato de intercambio en el sentido más amplio posible. Como formato de datos, la versión 2 de GML estándar, soporta la descripción de *features* simples, otro estándar OpenGIS. Esto significa que objetos (geometrías) en 0, 1 y 2 dimensiones y sus atributos pueden ser descritos usando GML 2 [Prins, 2003].

Como podemos ver GML favorece la interoperabilidad, el intercambio y manejo de datos, permitiendo un mayor control sobre ellos y al ser estándar, se apega a una



especificación y se evita el problema de heterogeneidad de formatos. Si el lector desea más referencias acerca de GML puede consultar [Cepeda, 2003].

2.1.3 *Features* (rasgos geográficos)

El consorcio OpenGIS [OGC, 2004] define a un *feature* geográfico como "la abstracción de un evento del mundo real que está asociado a una ubicación relativa con la Tierra" mediante un sistema de referencia espacial (SRS).

Una representación digital del mundo real puede ser pensada como un conjunto de *features*. El estado de un *feature* está definido por un conjunto de propiedades, donde cada propiedad puede pensarse como una tupla {nombre, tipo, valor}. Estas propiedades pueden ser estimadas geométricamente y el nombre y el tipo de cada propiedad está determinado por su definición de tipo, es decir, por el esquema de aplicación que lo norma.

Un *feature* geográfico es esencialmente una lista de nombres de propiedades. Algunas o todas esas propiedades pueden ser geo-espaciales, describiendo la posición y forma del *feature*. Cada *feature* tiene un tipo, el cual es equivalente a una clase en terminología de modelado de objetos. Tal definición de clase prescribe las propiedades que un *feature* particular debe tener de ese tipo. Por ejemplo, un Camino (*road*) puede ser definido para tener un nombre, una superficie de construcción, un destino, y una línea central. Las propiedades por sí mismas son modeladas en UML (*Unified Modeling Language*) como asociaciones, o como atributos, de la clase de *feature*. El tipo de propiedad del *feature* es dado por el rol de una asociación, o por el nombre del atributo. Los valores de las propiedades son por sí mismos también instancias de tipos o clases definidos. Así, el nombre *Road* (Camino) es una cadena de texto, la superficie de construcción puede ser un texto seleccionado de una lista enumerada, el destino es otro *feature* de tipo *Town*, y la línea central es un *LineString*, el cual es una propiedad geométrica [GML, 2002].

En UML una característica es representada como asociación o como atributo, aunque es común que una propiedad con un tipo complejo o un tipo altamente estructurado, sea modelada como una asociación, mientras que las propiedades simples son

comúnmente atributos de clase. Si el valor de una propiedad sólo existe en la presencia de un *feature*, tal como el nombre *Road*, entonces se puede usar la asociación de composición de UML o ser representado como un atributo, estos dos métodos son funcionalmente equivalentes. Sin embargo, si el valor de la propiedad está ligado débilmente al objeto y el valor de la propiedad es un objeto que puede existir independientemente del *feature*, tal como *Town* que es el destino de *Road*, entonces se debe usar una forma de asociación UML llamada agregación [GML, 2002] (ver Figura 2.3).

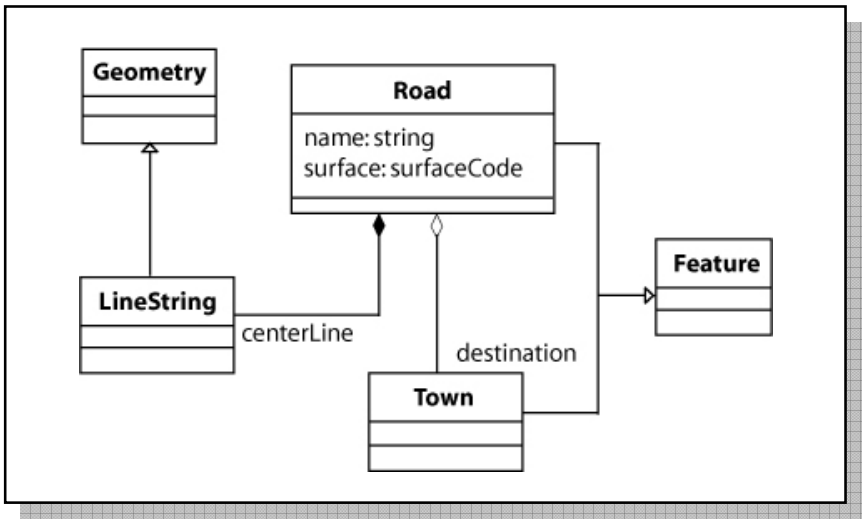


Figura 2.3 – Relaciones de composición y agregación

2.1.3.1 *Features* simples

Las geometrías de los *features* geográficos están restringidas a las que el consorcio OpenGIS llama geometrías simples.

El OGC define el concepto de *features* simples como "*features* cuyas propiedades geométricas están restringidas a geometrías 'simples' para las cuales, las coordenadas están definidas en dos dimensiones y la delineación de la curva, en el caso de que ésta exista, esta sujeta a la interpolación lineal" [OGC, 2001].

El modelo geométrico para *features* simples permite geometrías que son colecciones de otras geometrías (colecciones homogéneas de multi-puntos, multi-líneas y multi-polígonos, o colecciones heterogéneas de geometrías).

La figura 2.4 [GML, 2002] es el modelo de objetos geométricos para *features* simples, el cual tiene una clase base abstracta *Geometry* y asocia cada objeto geométrico con el sistema de referencia espacial (SRS) que describe el espacio coordenado en el cual se define el objeto. Por ejemplo, tomemos la geometría *Point* de este modelo, podemos ver que más de un punto puede formar una geometría más compleja *MultiPoint*, y de manera análoga se puede decir que dos o más puntos pueden formar una geometría *LineString*. De manera similar se interpretan todas las demás geometrías de este modelo las cuales tienen en común que todas heredan de la clase abstracta *Geometry* la cual, por lo tanto, asocia a cada geometría a un lugar específico de la tierra mediante el SRS.

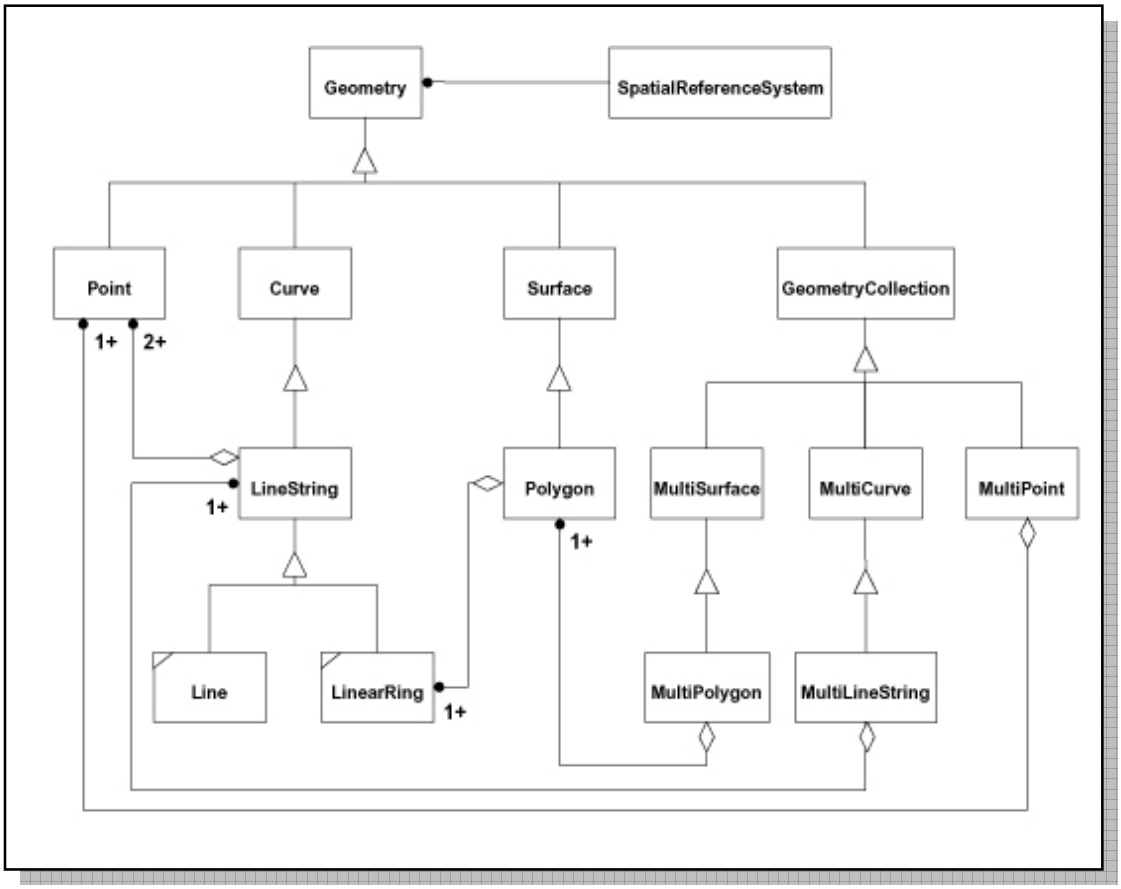
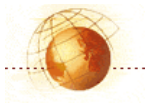


Figura 2.4 – Modelo geométrico para *features* simples



2.2 Tecnologías de Ingeniería de *Software*

En esta sección del documento veremos los conceptos de ingeniería de software involucrados en el desarrollo de esta tesis. Se divide en las secciones: Ingeniería inversa y *Unified Modeling Language*.

2.2.1 Ingeniería inversa

El análisis de código fuente ha motivado un interés creciente desde los primeros años del auge del desarrollo de software. Analizar código fuente de forma automática, o al menos asistida, nos permite entre otras cosas, apoyar varias áreas de marcada importancia como son: el mantenimiento de sistemas, la integración de código legado en nuevas aplicaciones, la recolección de medidas para evaluar la calidad del software desarrollado, la transformación de programas, la refabricación de software (que puede verse como caso especial de transformación de programas), entre otras.

Una de las áreas de especial interés actualmente, y que se apoyan fundamentalmente en el análisis de código es la ingeniería inversa, y lo que la complementa: la reingeniería. Ingeniería inversa, como su nombre indica, hace referencia al procedimiento contrario al que se realiza habitualmente en lo que denominamos Ingeniería del Software.

En Ingeniería del Software el objetivo es ir modelando la realidad en pasos sucesivos en los que se baja el nivel de abstracción, hasta llegar a una representación computacional que resuelva el problema planteado. De ahí que los "documentos iniciales" sean especificaciones de requisitos en lenguaje natural (o algún lenguaje más formal de muy alto nivel), modelos conceptuales expresados en lenguajes de modelado (como es el caso de UML que se trata en la siguiente sección), diseños de arquitectura y modelos de diseño detallado hasta que finalmente se tenga una implementación escrita en un lenguaje de programación, y por lo tanto ejecutable. En Ingeniería del Software, lo ideal sería contar con el soporte necesario para que, partiendo de un lenguaje de muy alto nivel de abstracción, se pueda generar, lo más automáticamente posible, una salida que sea el programa que lo ejecuta. El

procesamiento iría, al igual que en los compiladores, de mayor nivel a menor nivel de abstracción.

En cambio, la Ingeniería inversa intenta extraer modelos del dominio de la solución y, más aún, del dominio del problema a partir del código final. De ahí que se realice un procesamiento en el que, a diferencia de los compiladores, el código final es de un nivel de abstracción mayor que el código fuente. En este caso, el lenguaje del código final es un lenguaje de modelado, o una abstracción de un lenguaje de modelado, y el lenguaje del código fuente es un lenguaje de programación. El objetivo último de la ingeniería inversa consiste en separar de los detalles de implementación la información esencial del problema y de su solución. De esta forma, por ejemplo, se suplen las carencias de documentación cuando se trabaja con código legado o se puede cerrar el ciclo mediante lo que se conoce como reingeniería. La reingeniería pasa por la aplicación de una nueva iteración de un proceso de Ingeniería del Software en el que se tomen nuevas decisiones de diseño, cambiando hasta de paradigma de desarrollo si fuera el caso. Todo esto a partir de la información extraída mediante la aplicación de ingeniería inversa al código fuente de un sistema previamente desarrollado.

Todo el proceso de ingeniería inversa y su implementación se trata a profundidad en el capítulo 4 de este documento de tesis.

2.2.2 *Unified Modeling Language* (UML)

El lenguaje de modelado unificado (UML por sus siglas en inglés) es un lenguaje gráfico para la visualización, especificación, construcción, y documentación de cada una de las partes que comprende el desarrollo de software. UML proporciona una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un leguaje de programación determinado, esquemas de bases de datos y componentes de software reusables.

[Booch, 1999] define un modelo como “una simplificación de la realidad” el cual construimos para poder entender mejor el sistema que estamos desarrollando. A través del modelado, se logran 4 objetivos:

- Los modelos ayudan a visualizar un sistema como es o como queremos que sea.
- Los modelos permiten especificar la estructura o comportamiento de un sistema.
- Los modelos proporcionan una plantilla que sirve de guía en la construcción de un sistema.
- Los modelos documentan las decisiones que hemos tomado.

Es definitivamente cierto que entre más grande y complejo es un sistema, más importante se vuelve modelar, por una simple razón: “Construimos modelos de sistemas complejos porque no podemos comprender tales sistemas en su totalidad”.

La habilidad humana tiene límites para entender la complejidad. A través del modelado, hacemos más pequeño el problema de estudio, enfocándonos sólo a un aspecto a la vez. Ese es esencialmente el enfoque de “divide y venceras” que Edsger Dijkstra dijo hace unos años: “Ataca un problema difícil dividiéndolo en una serie de problemas más pequeños que puedas resolver”. Además, a través del modelado se amplifica el intelecto del hombre. Un modelo escogido apropiadamente puede lograr que el modelador trabaje a niveles mayores de abstracción [Booch, 1999].

La figura 2.5 es un ejemplo de un diagrama de clase UML, en donde se muestra la clase *WFSRequest*. Podemos ver que hereda de la clase *Request*, que importa la clase *String*, y que el paquete *org.vfny.geoserver.requests.wfs* hereda de ésta clase. También se pueden apreciar los métodos y atributos propios de la clase.

UML es apropiado para modelar sistemas no importa cual sea el giro. Es un lenguaje que no es difícil de entender o de usar. Según [Booch, 1999] aprender UML efectivamente comienza con la formación de modelos conceptuales del lenguaje, lo cuales requieren de tres elementos principales:

- Los bloques de construcción básicos de UML

- Las reglas que dictan como deben de juntarse esos bloques de construcción y
- Algún mecanismo común que se aplica a través del lenguaje.

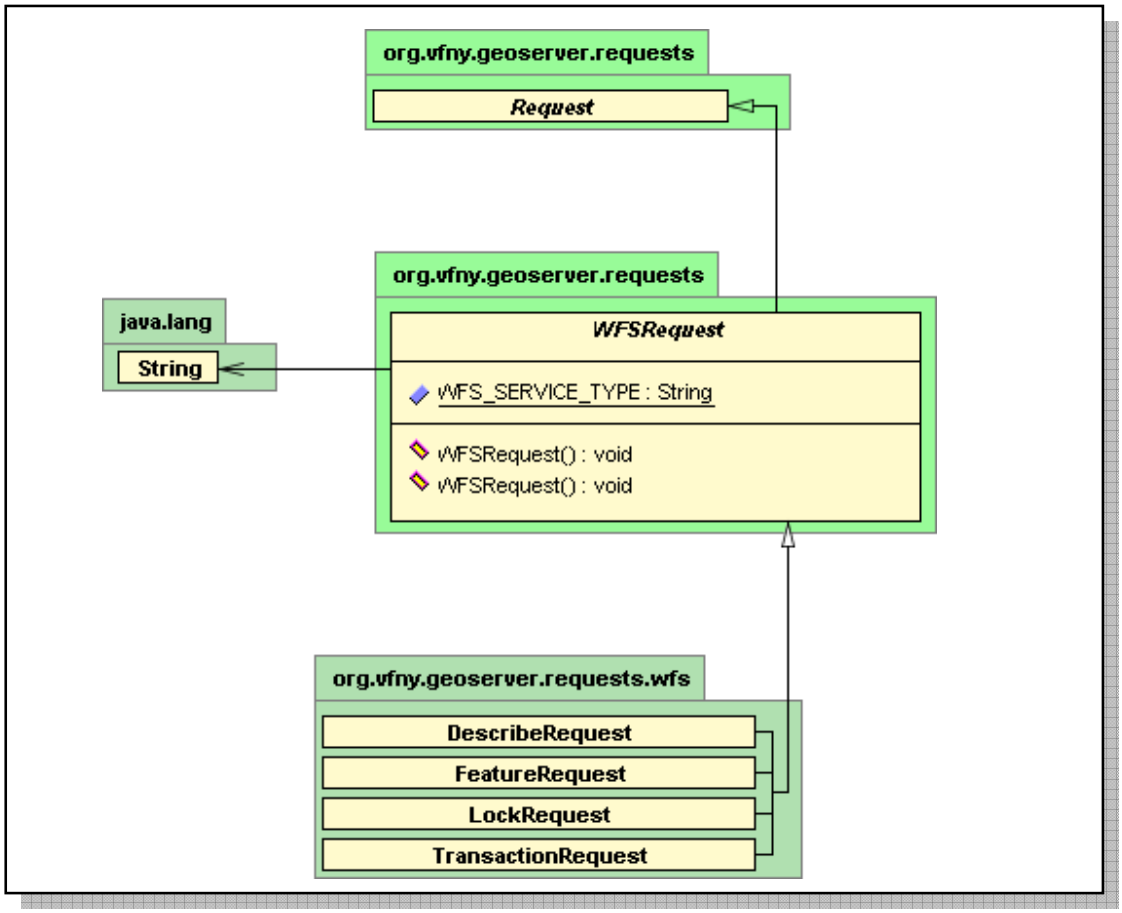


Figura 2.5 – Ejemplo Diagrama de clase UML

2.2.2.1 Principios del modelado

El uso del modelado tiene una rica historia en todas las disciplinas de la ingeniería. Esta experiencia sugiere 4 principios básicos para el modelado [Booch, 1999].

- **Primero** – *La elección de que modelos crear tiene una influencia profunda en como atacar un problema y como formar una solución.* Esto quiere decir que debemos escoger bien los modelos. Los modelos correctos permitirán representar los problemas de desarrollo más difíciles, ofreciéndonos la visión que de otra manera no hubiéramos podido comprender; los modelos

equivocados nos guiarán erróneamente, causando que nuestra atención se concentre en cosas irrelevantes.

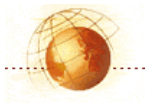
- **Segundo** – *Cada modelo puede ser expresado a diferentes niveles de precisión*. Los mejores modelos son aquellos que nos dejan escoger el grado de detalle, dependiendo de quien está viendo el modelo y porque necesita verlo. Por ejemplo, un analista o un usuario se enfocan en cosas como el ¿Qué?, mientras que el desarrollador se enfoca en cosas como el ¿Cómo? De cualquier manera el modelo debe poder permitir ver al sistema en diferentes niveles de detalle en ocasiones distintas.

- **Tercero** – *Los mejores modelos están conectados a la realidad*. Es mejor tener modelos que tienen una clara conexión con la realidad, y en donde la conexión sea débil, saber exactamente como esos modelos están divorciados del mundo real. Todos los modelos son simplificaciones de la realidad, el truco es estar seguros que las simplificaciones no oculte detalles importantes.

- **Cuarto** – *Ningún modelo sencillo es suficiente. Los sistemas que no son triviales se enfocan mejor a través de un pequeño conjunto de modelos independientes ligados*. Se pueden tener modelos que se pueden construir y estudiar por separado, pero éstos están interrelacionados. Por ejemplo, en el caso de un edificio, se puede estudiar los planos eléctricos por separado, pero podemos ver su mapeo en los planos de piso, y quizá hasta su interacción con la tubería en el plano de plomería.

2.2.2.2 UML es un lenguaje.

UML es sólo un lenguaje y, por lo tanto, es sólo una parte del método de desarrollo de software. Un lenguaje provee un vocabulario y las reglas de combinación de palabras en ese vocabulario para lograr la comunicación. Un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se enfocan en la representación conceptual y física de un sistema. Un lenguaje de modelado como UML es un lenguaje estándar para planear un software.



[Booch, 1999] explica que UML es un leguaje para:

- **Visualizar**
- **Especificar**
- **Construir**
- **Documentar**

Para entender UML se necesita formar un modelo conceptual del lenguaje. Cuando se vaya adquiriendo experiencia en UML, se podrán construir modelos conceptuales usando características avanzadas de este lenguaje. La figura 2.6 muestra el vocabulario que abarca UML. Si se desea mayor referencia acerca de UML se pueden consultar [Booch, 1999].

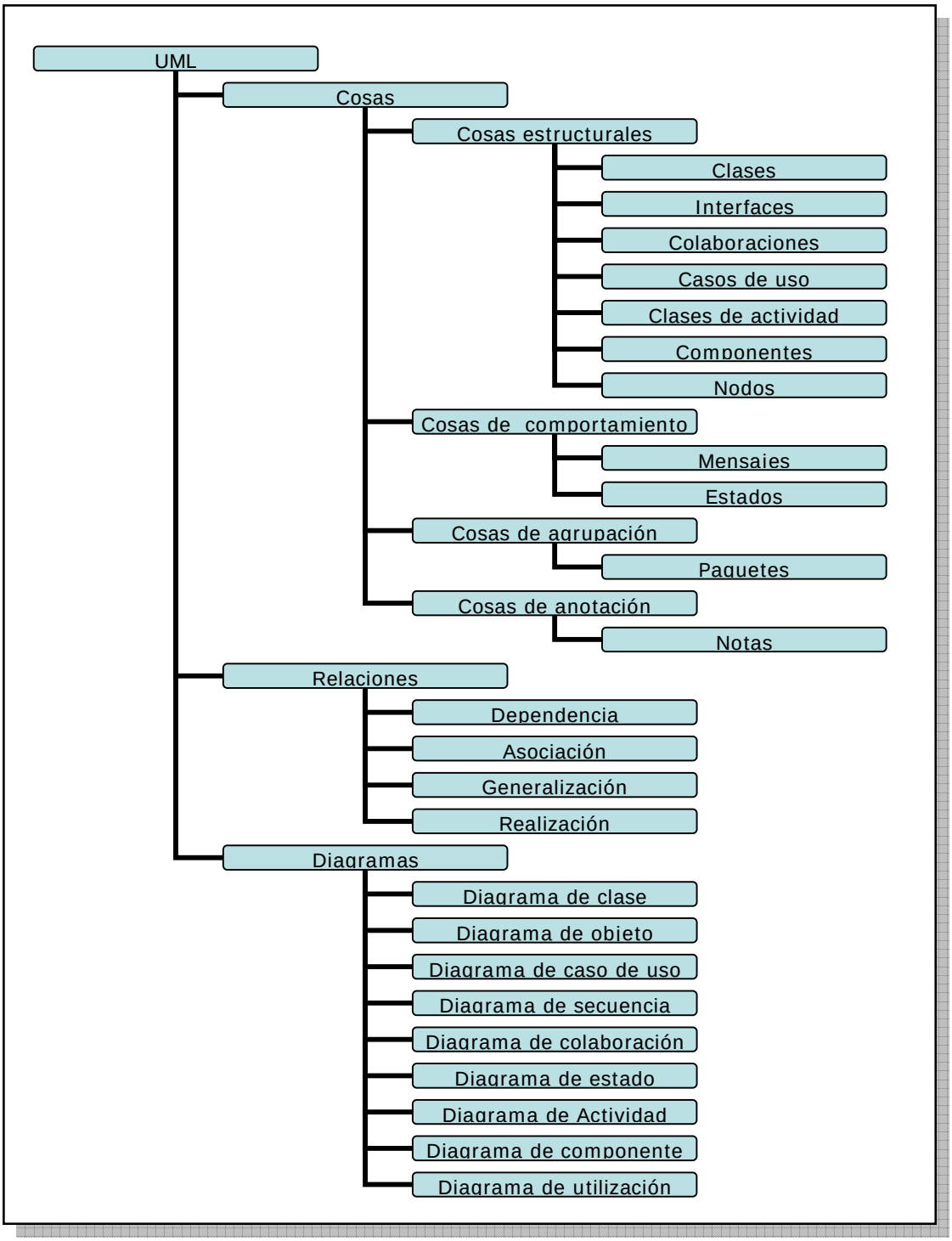
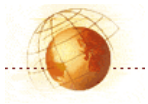


Figura 2.6 – Vocabulario UML



2.3 Tecnologías *web*

Esta sección abarca las tecnologías *web* que sirvieron de base para el desarrollo de este proyecto de tesis y particularmente se presentan los conceptos: *Software* distribuido, *Web Service*, XML *Web Service* y finalmente el tema de mayor importancia para este proyecto de tesis, *Web Feature Service*.

2.3.1 *Software* distribuido

La finalidad de Internet es prestar servicio a las comunidades, los accionistas, los clientes y los empleados. Cada vez con mayor frecuencia, estos servicios se prestan a las comunidades a través de funciones de software que se han migrado a la red.

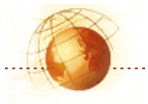
El software puede estar distribuido, guardado en componentes en los directorios de un sistema distribuido, y ser solicitados para realizar funciones específicas. Puede incluir aplicaciones que involucren diferentes enfoques de distribución, pero también puede contener diferentes innovaciones en aspectos como *Web Services* u otros modelos de software por componentes.

Web Services es computación distribuida rápida y eficiente empleando estándares abiertos como XML y HTTP para implementar la invocación remota de métodos entre aplicaciones.

2.3.2 Web Service

Un *Web Service* es un componente de software que presenta un conjunto de funcionalidades que sigue una arquitectura bien definida y puede ser invocada por otra aplicación, para formar parte de un proceso de negocio u otro servicio, haciendo uso de los estándares del Internet (XML, HTML, XHTML, HTTP, entre otros).

Los *Web Services* son componentes aplicativos auto-descriptivos que pueden descubrir y emplear otros componentes aplicativos estandarizados para llevar a cabo tareas complejas vía las tecnologías del Internet. Representan un nuevo esquema de integración y cómputo distribuido, ya que al ser componentes de *software* expuestos



vía *web*, se convierten en un elemento clave para crear arquitecturas orientadas a servicios.

Los *Web Services* son una forma estándar de interactuar con aplicaciones de negocios a través del *web*, por lo cual representan el futuro del Internet. Consisten en formas estandarizadas que poco a poco están siendo adoptadas en el mercado para:

- Transportar componentes universalmente (IP, HTTP, SMTP);
- Compartir contenido y objetos dispares (XML, Java);
- Describir un *Web Service* para uso específico (WSDL);
- Comunicarse entre *Web Services* (SOAP, JMS);
- Publicar y localizar determinados *Web Services* (UDDI).

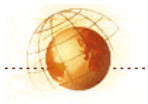
Los *Web Services* provienen de:

- La necesidad de intercambio de información y transacciones de manera automatizada, vía el Internet;
- Los estándares de Java, XML y COM;
- La evolución del EDI (*Electronic Data Interchange*) hacia los marcos de referencia tipo ebXML (*e-business with* XML);
- La evolución de aplicaciones aisladas a servicios modulares, diferenciados y gestionados;
- La necesidad de una plataforma para la evolución futura a procesos de negocios estandarizados por industrias verticales (vocabularios comunes, procesos homologados, uso de estándares de mensajería, *trading partner agremeents*, etc.).

Los *Web Services* son una de las nuevas tecnologías más prometedoras para las empresas. Los ejecutivos visualizan a los *Web Services* como el siguiente paso en sus estrategias de e-Business y los desarrolladores cada vez incorporan más componentes de *Web Services* en proyectos de *software*.

Los *Web Services* prometen una metodología neutral en cuanto a la plataforma y basada en:

- estándares para la integración de aplicaciones



- automatización de procesos de negocio
- publicación/consumo de información de diversas fuentes
- intercambio empresarial.

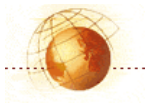
Esto se debe a que los *Web Services* alcanzarán dos objetivos tecnológicos altamente deseados de manera anticipada:

- Interfaces de integración ligera de sistema a sistema basadas en estándares;
- Componentes independientes y reutilizables para el desarrollo rápido de aplicaciones y servicios.

Estas tecnologías son importantes para alcanzar una arquitectura orientada a servicios, en la cual diversos componentes de software heterogéneos puedan describirse a sí mismos e interactuar para entregar servicios conforme sean requeridos. Esta arquitectura también representa la clave para la empresa de "tiempo real", un modelo de negocio avanzando, en el cual la ventaja competitiva se obtiene a través del acceso inmediato a la información y a las transacciones operativas [QoS, 2003].

En [Brunner, 2002], Robert desglosa la pregunta ¿Qué son los *Web Services*? y llega a la siguiente definición: los *Web Services* son una solución para proveer comunicación de aplicación a aplicación sobre Internet.

Los *Web Services* ofrecen una flexibilidad y eficiencia sin precedente en la integración y desarrollo de aplicaciones corporativas. De cualquier manera, este poder se obtiene a través de una nueva y compleja arquitectura de aplicaciones. Las soluciones tradicionales para la gestión de aplicaciones no fueron diseñadas para administrar componentes heterogéneos de *Web Services*, ni para el manejo de la dinámica de ejecución tan particular de los mismos. Para poder comprender el verdadero y enorme potencial de los *Web Services*, es necesario contar con una visión diferente sobre la gestión de servicios administrados o *Web Services*. Para poder implementarlos, es necesario crear una cultura de migración de las aplicaciones cliente-servidor a tres capas [QoS, 2003].



2.3.3 XML *Web Services*

Con el auge de todas estas tecnologías surge un nuevo tipo de servicios vía *web*, los XML *web services*. El termino XML *Web Services* describe una manera estandarizada de integrar aplicaciones *web* usando XML (eXtensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web Services Definition Service) y UDDI (Universal Description Discovery and Integration) estándares abiertos sobre Protocolos de Internet.

XML se utiliza para marcar los datos con etiquetas, SOAP se utiliza para transferir los datos, WSDL se utiliza para describir los servicios disponibles y UDDI se utiliza para publicar qué servicios están disponibles. Se utilizan principalmente como medios para comunicar negocios de unos con otros y con los clientes, los XML *Web Services* permiten que las organizaciones se comuniquen sin un conocimiento detallado de los sistemas de Información de cada una.

Diferente a los sistemas cliente/servidor y/o sistemas Web, los XML *Web Services* no proveen una interfaz gráfica al usuario, sino que comparten la lógica de negocios, junto con los procesos y datos por medio de una interfaz de programa utilizando una red. Las interfaces se interconectan de manera transparente, por lo tanto, los clientes nunca lo notan.

Los XML *Web Services* permiten que las aplicaciones compartan información y que además invoquen funciones de otras aplicaciones, independientemente de cual sea el sistema operativo o la plataforma en que se ejecutan y cuáles sean los dispositivos utilizados para obtener acceso a ellas. Aunque los XML *Web Services* son independientes entre sí, pueden vincularse y formar un grupo de colaboración para realizar una tarea determinada [Bankhacker, 2003]. Esta última definición nos introduce al contexto de los sistemas distribuidos.

2.3.4 *Web Feature Services*

Un caso particular de todas las tecnologías mencionadas anteriormente aplicadas al

campo de los sistemas de información geográficos (SIG) son los servidores de rasgos geográficos o WFS.

Durante la iniciativa OpenGIS Consortium (OGC) *Web Mapping Test* (WMT1), se desarrollaron dos especificaciones:

- OpenGIS© Web *Map Service Implementation Specification*
- OpenGIS© *Geography Markup Language* (GML) 2.0 *Implementation Specification*.

El documento que define a los WFS, surge como un siguiente paso a esta iniciativa y propone interfaces para describir operaciones de manipulación de datos sobre *features* geográficos usando HTTP como plataforma de cómputo distribuido.

Como ya vimos existe una gran variedad de *Web Services* disponibles actualmente en la *web*. La arquitectura WFS**,** permite a un cliente recuperar datos geoespaciales codificados en GML de múltiples servidores de *features*. GML como ya vimos, es una extensión de la codificación XML, que se usa para la transportación y almacenamiento de geometrías y propiedades de *features* o rasgos geográficos. La figura 2.7 [WFS, 2002] muestra la interface de un WFS. Ahí podemos ver que al cliente no le interesa dónde o cómo se encuentre la información en las fuentes de datos, solo le interesa saber donde se encuentra el servicio al cual se va a conectar para enviar y recibir peticiones.

Algunas de las características que un WFS debe tener son:

- Las interfaces deben ser definidas en XML.
- GML debe ser usado para expresar *features* en la interface.
- Como mínimo un WFS debe poder presentar *features* usando GML.

Un WFS soporta operaciones como: *INSERT, UPDATE, QUERY* y *DISCOVERY* sobre *features* geográficos, usando HTTP como la plataforma de cómputo distribuido.

Es la función de un WFS, en su interacción con el sistema de almacenamiento de información usado, administrar *features*, para asegurarse que los cambios a los datos son consistentes.

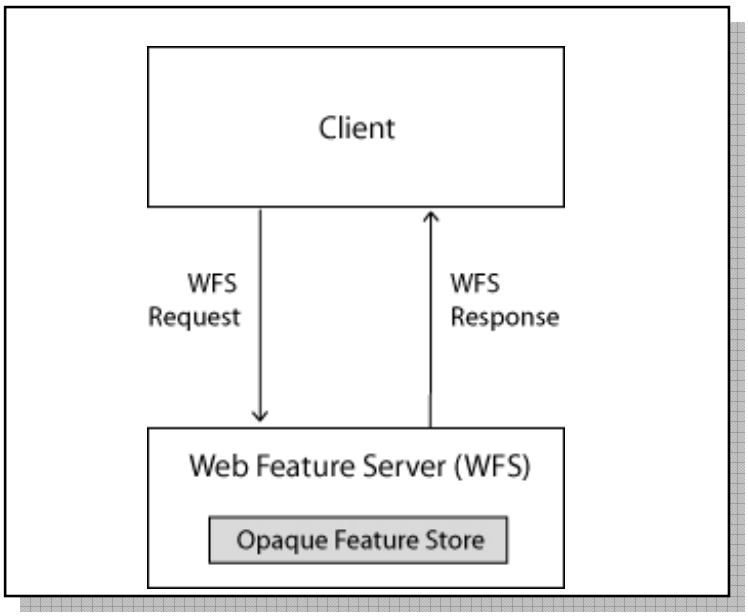
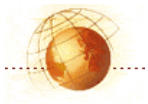


Figura 2.7 – Interface de un *Web Feature Service*

Dependiendo del soporte que tenga un WFS sobre estas operaciones, éstos se clasifican en:

- ***Basic* WFS** - Web Feature Service de sólo lectura (*Read-Only*). Un WFS básico implementa las operaciones *GetCapabilities*, *DescribeFeatureType* y *GetFeature*.
- ***Transaction* WFS** – *Web Feature Service* Transaccional (*Transactional*). Un WFS transaccional soporta todas las operaciones del WFS básico, además, implementa las operaciones de transacción, como son *Insert*, *Update* y *Discovery*. Opcionalmente, un WFS transaccional puede implementar la operación *LockFeature*.

Un WFS no se define ni se genera en base a un dominio específico, es decir, en un WFS se puede cargar cualquier cartografía que cuente con las características definidas por el mismo, no se implementa de manera particular para satisfacer a una sola cartografía. Dentro de las características que definen a la cartografía podemos mencionar la escala y el sistema de referencia usado, siendo estas dos las mas importantes. Todo lo referente al servicio WFS se puede consultar a detalle en el apéndice D.



2.4 Justificación

Como pudimos ver, en este capítulo se trataron varios conceptos interesantes e importantes para el contexto de este proyecto de tesis, pues todos ellos al combinarse contribuyeron en su desarrollo. Ahora que se han explicado estos conceptos podemos justificar su uso en este proyecto de tesis. Este *Web Feature Service* es un SIG que se logró utilizando tecnologías abiertas para su implementación, como por ejemplo GML para el almacenamiento y transporte de la información geográfica y está implementado de acuerdo a una especificación de implementación definida por el consorcio OpenGIS. Para hacer posible esta implementación fue necesario usar la ingeniería inversa que es una parte del proceso de reingeniería del software para obtener información necesaria de una implementación pública que carecía de ella, con el fin de lograr una correcta implementación. Es por esto que son muy importantes todos y cada uno de los conceptos explicados en este capítulo. En el siguiente capítulo trataremos algunos proyectos que ya han sido desarrollados y que utilizan algunos de estos conceptos y que además fueron en parte inspiración para este proyecto de tesis.