UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION

UN FRAMEWORK PARA APLICACIONES WEB DE MAPAS

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL EN COMPUTACION

CRISTIAN MAURICIO OLATE OPAZO

PROFESOR GUIA:
MARIA CECILIA BASTARRICA PIÑEYRO

MIEMBROS DE LA COMISIÓN:
ERIC PIERRE TANTER
PABLO BARCELO BAEZA

SANTIAGO DE CHILE
AÑO 2012

Resumen

Casi al final de la cadena de administración de la información geográfica en las organizaciones, se encuentra la tarea de la publicación. Para la automatización de dicha tarea, existe en el mercado una amplia variedad de *software* GIS, dentro de la cual destaca la oferta de ESRI. Sin embargo, los costos de implementar una solución como la ofrecida por ESRI no pueden ser asumidos por pequeñas y medianas organizaciones. De esta situación se hace cargo este trabajo.

Este trabajo consistió en la construcción de un marco de desarrollo base para publicar información geográfica, el cual consta de dos elementos principales:

- Un conjunto de librerías que extienden un *framework* de desarrollo *web*, para el trabajo con objetos geográficamente referenciados.
- Una aplicación *web*, construida sobre la base del *framework* extendido, para generar y administrar Aplicaciones Web de Mapas.

El resultado obtenido, basado en tecnologías gratuitas y de código abierto, fue validado a partir de la implementación de un caso de uso, consistente en la creación y extensión de una Aplicación Web de Mapas en un dominio específico.

Se concluye que el trabajo realizado permitirá a las organizaciones de menor tamaño publicar la información geográfica de la que disponen de manera controlada, y se plantean finalmente algunas recomendaciones sobre cómo seguir extendiendo las funcionalidades del marco de desarrollo base.

A Baldomero, que gustaba de mis relatos.

Agradecimientos

Debo partir agradeciendo a la profesora Cecilia Bastarrica por acoger y guiar mi proyecto; espero haber llenado sus espectativas.

Gracias a todos los que hicieron más fácil mi larga estadía en la Facultad. Por la amistad, ya fuese por periodos o hasta el día de hoy, agradezco a Mauricio, Sergio y Guillermo (Q.E.P.D.). Por el apoyo y comprensión, agradezco a la gente de Bienestar. Agradezco también a la Facultad entera, con la cual tengo una enorme deuda de gratitud.

Gracias a Geneviève, la mejor jefa que pude tener; nunca perdió la fe en que respondería a su pregunta: *¿Y, cuándo?*.

Gracias a Perla, por el apoyo y aliento.

Gracias a Marcelo, por la disposición a ayudar.

Gracias a la Liga del Mal (Vale, Karla y Sergio), por los buenos momentos.

Finalmente, agradezco a Andrea, por la voluntad para esperarme por tanto, por seguirme, por volver. Gracias por salvar mi día, todos los días.

Índice General

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción.

1.1. Introducción a los GIS.

Un Sistema de Información Geográfica (en adelante GIS, por sus siglas en inglés) es una tecnología horizontal que involucra una gran variedad de usos y aplicaciones en distintos ámbitos de la industria. Formalmente un GIS se define como la integración de *hardware*, *software*, recursos humanos y procedimientos, destinados a la captura, almacenamiento, ajuste, manipulación, análisis y representación de datos espacialmente referenciados sobre la tierra en forma gráfica y alfanumérica[1] [1]. La Figura 1.1 ilustra el modelo de un GIS como herramienta para convertir datos espaciales en información útil a través del análisis [2], y cómo el retorno de la inversión llega al momento de producir salidas en forma de mapas.

Un GIS sirve para resolver problemas complejos de gestión y planificación en un amplio espectro de disciplinas, respondiendo a diversos tipos de preguntas, tales como [3]:

- Qué hay en un territorio: qué tipo de cultivos, qué vegetación, cuánta población, cuántos clientes de mi empresa, etc.
- Dónde ocurre o ha ocurrido un hecho concreto: actividad criminal, fenómenos climáticos tales como como huracanes, heladas, niveles de contaminación, etc.
- Qué distribución espacial tienen los fenómenos: contaminación por ozono, riesgos de derrumbre e incendios, pandemias, etc.

[1] En el alcance de este trabajo, sin embargo, entenderemos por GIS el conjunto de herramientas tecnológicas destinadas a la gestión y despliegue gráfico de los datos espacialmente referenciados.

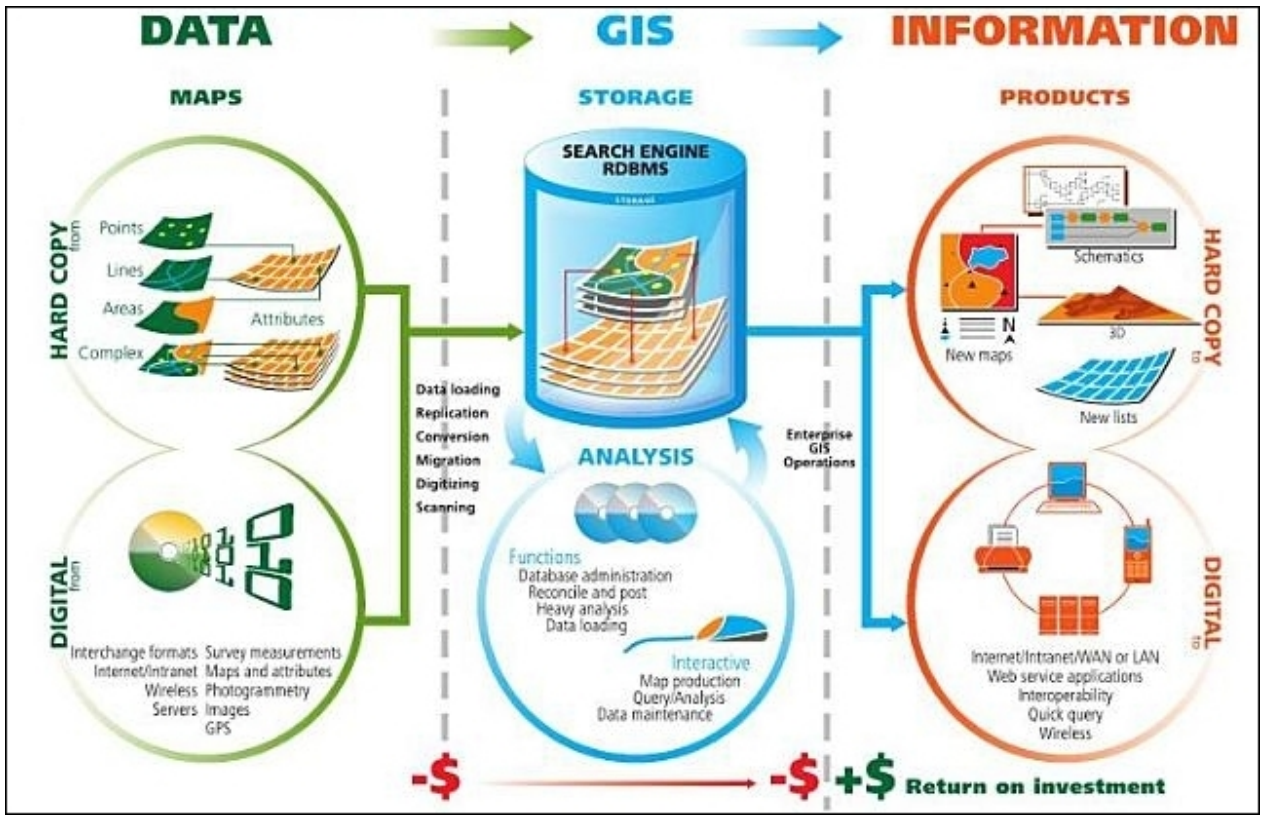

Figura 1.1: Componentes de un GIS.

- Qué tendencias temporales se dan en un territorio: cómo evoluciona el espacio urbano, cómo han crecido ciertos sectores de población, cómo ha avanzado una exploración minera.

- Qué rutas decidir para desplazamientos: logística para repartos, trayectos más cortos, rutas de interés turístico.

- Qué tal si: modelamiento de escenarios a partir de datos geográficos, impacto ambiental, etc.

Las aplicaciones de un GIS son transversales a las actividades humanas, incluyendo la industria minera, planificación urbana, *marketing*, arqueología, logística, turismo, entre muchas otras; y sirven a distintos tipos de profesionales en una organización. De ahí el valor que adquiere la implementación de un GIS[2]. Sin embargo, el acceso a este tipo de herramientas es limitado, pues los clientes deben equiparse con *software* profesional, generalmente de al-

[2]Aplicaciones como *Google Earth* han hecho visible de modo muy efectivo el valor de la información georeferenciada.

to costo monetario, además de una infraestructura de *hardware* de alto rendimiento [4]. Es aquí donde nace el espacio para un nuevo tipo de herramientas GIS: las Web GIS.

1.2. Un framework para Aplicaciones Web de Mapas.

Las Web GIS son aplicaciones GIS basadas en *web*. Dado que las tareas relativas al procesamiento de los datos son realizadas en un servidor, y que para su uso basta con un navegador *web*, el costo de procesamiento de datos que asumen los clientes es muy bajo en comparación con las aplicaciones GIS clásicas de escritorio, siendo por lo tanto más accesibles. Este nicho ha sido explotado de un modo creciente en los últimos años, multiplicándose las aplicaciones *web* de mapas (en [5] y [6] se puede revisar una amplia lista) y surgiendo iniciativas incluso a nivel de gobierno como la de GeoEuskadi [7].

Para el desarrollo de aplicaciones Web GIS existen en la actualidad variadas herramientas. Por el lado de las herramientas de código libre, existen extensiones para *frameworks* de desarrollo como Django (Python) y Ruby on Rails (Ruby), que permiten el desarrollo ágil de Aplicaciones Web de Mapas. Por el lado de las herramientas propietarias, la más destacable es el Web ADF[3] de ESRI [8], que permite el desarrollo, con apenas un par de configuraciones básicas, de una aplicación *web* que toma como *input* mapas publicados por ArcGIS Server, con funcionalidades tipo como *zoom*, cálculo de distancias, consultas espaciales y geoprocesamiento, entre otras.

La meta de este trabajo es la construcción de un *framework* para desarrollo de Aplicaciones Web de Mapas, de código libre y basado en herramientas de código libre también, con funcionalidades inspiradas en las herramientas propietarias como la de ESRI[4]. La motivación principal para hacer esto, es poner al alcance de pequeñas y medianas organizaciones las herramientas necesarias para gestionar su información geográfica.

[3] *Application Development Framework*.

[4] Se hace referencia aquí a las herramientas para la generación y administración de aplicaciones *web* de mapas proporcionadas por ESRI, en particular a ArcGIS Server (servidor de mapas) y al SDK para desarrollo de aplicaciones *web* de mapas en ArcGIS Server.

1.3. Justificación.

El valor comercial de las licencias de *software* GIS, tanto de escritorio como *web* es alto. El Cuadro 1.1 ilustra los costos de licencias para los productos de ESRI necesarios para implementar un GIS en una organización de tamaño medio, que permita tanto la producción como la publicación de información geográfica en *web*.

Software GIS	**Costo (MM $)**
ArcGIS Desktop (ArcInfo 9.3.1)	9.8
ArcGIS Server Standard (4 core)	17.9

Cuadro 1.1: Costos licencias ArcGIS.

Tales costos son sólo de licencias, no incluyendo por tanto el resto de las herramientas necesarias para implementar un GIS en una organización, a saber: *hardware*, capacitación y asesoría. Resulta entonces demasiado costoso, en la mayor parte de las pequeñas empresas, montar una infraestructura de *hardware* y *software* necesarias para soportar un GIS como el ofrecido por ESRI.

Un caso que ejemplifica esta situación es el de las municipalidades. En este tipo de organizaciones, la información geográfica es especialmente relevante en diversas áreas: plan regulador de la comuna, gestión de obras viales, estacionamientos municipales, alumbrado público, colectores de aguas o gestión territorial en general. La capacidad para generar y administrar esta información, y canalizarla después hacia quienes interese, es una necesidad latente que, dados los costos de una implementación de ArcGIS, no ha podido ser resuelta por completo. Una evidencia concreta de la falta de oferta que satisfaga estas necesidades a costo razonable, son las numerosas licitaciones públicas para implementación de un GIS, algunas declaradas desiertas[5].

Surge aquí la alternativa de las herramientas de código abierto. Pero ni los generadores de Aplicaciones Web de Mapas, como Chameleon o Mapfish, ni menos las extensiones espaciales a *frameworks* de desarrollo *web*, como GeoDjango o GeoRuby, logran acercarse a la funcionalidad ofrecida por ArcGIS Server y su Web ADF. La principal piedra de tope es la ausencia

[5]Por ejemplo, la licitación pública 5482-154-LP09: Implementación de un Servicio de Información Geográfica, para la Municipalidad de Ñuñoa, en Mercado Público (`http://www.mercadopublico.cl`).

de una aplicación web que genere de modo simple Aplicaciones Web de Mapas extensibles, tal como lo hace ArcGIS Server Manager.

Por lo anteriormente expuesto, resulta atractivo contar con una aplicación basada en tecnologías de código abierto, que permita generar Aplicaciones Web de Mapas, extensibles, a partir de simples configuraciones. Ésto, tanto para las organizaciones que necesitan de herramientas para publicar la información geográfica de la cual disponen, como para quienes desarrollan aplicaciones de dominio específico; en ambos casos, el costo justifica el producto de este trabajo.

1.4. Objetivos.

1.4.1. Objetivo general.

El objetivo general de este trabajo es diseñar e implementar un marco de trabajo base para crear, de modo simple, Aplicaciones Web de Mapas extensibles. Tal desarrollo debe basarse en tecnologías de fuente abierta.

1.4.2. Objetivos específicos.

Los objetivos específicos pueden enumerarse como sigue:

- Implementar las extensiones geográficas al *framework* de desarrollo *web* CodeIgniter[6], necesarias para el trabajo con datos espaciales.
- Implementación de librerías destinadas al desarrollo de Aplicaciones Web de Mapas.
- Desarrollar una aplicación web para la generación y administración de Aplicaciones Web de Mapas extensibles.

Estos objetivos serán validados a través de una historia de uso que considere el proceso completo de manejo de la información geográfica, desde su organización hasta su publicación en la forma de una Aplicación Web de Mapas, extensible a un dominio específico.

[6]Una descripción de este *framework* y sus cualidades, pueden encontrarse en la Sección 2.3.

Capítulo 2

Antecedentes.

En este capítulo se introducen brevemente algunos de los conceptos básicos relacionados con las Geociencias y los estándares relativos al manejo de la información geoespacial. Éstos serán necesarios para comprender el dominio en el cual se desenvuelven las aplicaciones desarrolladas en este trabajo, así como el resto de este documento.

Se describe también la oferta de aplicaciones GIS existentes en el mercado, en particular las ofrecidas por ESRI. Puesto que el foco de este trabajo es igualar las funcionalidades para la publicación de Aplicaciones Web de Mapas ofrecidas por las herramientas de ESRI, se centra la descripción en ellas.

Se introducen, finalmente, las tecnologías de código abierto en las cuales se basan los desarrollos realizados, incluyendo una breve descripción de CodeIgniter, el *framework* de desarrollo elegido para ser extendido.

2.1. Conceptos básicos de Geografía y Geomática.

2.1.1. Geoide, elipsoides, datum.

No es del todo preciso decir que la Tierra es una esfera o una elipse; se define el término de *geoide*, que hace referencia a la superficie formada por todos los puntos en la tierra con igual gravedad, para describir una forma un poco más exacta del planeta (ver Figura 2.1, que muestra un modelo que ilustra de manera más realista la forma del planeta).

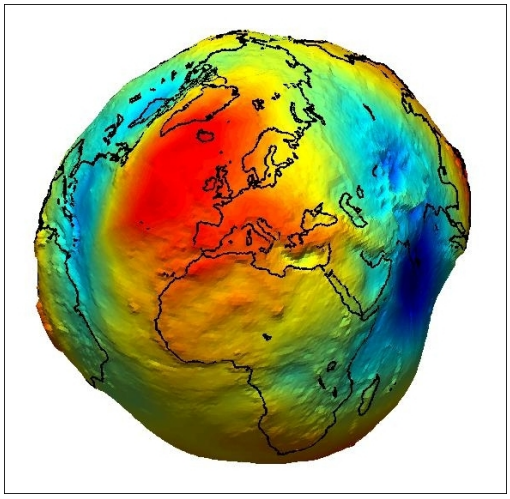

Figura 2.1: Modelo del geoide terrestre.

Definición 2.1.1 (*Geoide*)

Sea $g : \mathbb{R}^3 \to \mathbb{R}$ la función que a cada punto en la superficie terrestre le asigna el valor de la gravedad medida en ese punto. Se define el geoide terrestre como el conjunto

$$G = \{p \in \mathbb{R}^3 : g(p) = c\}$$

donde c es una constante.

En términos generales, se establecen los modelos de geoides a partir del campo gravitacional y tomando los puntos más cercanos a la superficie del nivel del mar.

Este es el motivo principal por el cual el problema del posicionamiento sobre la superficie terrestre no resulta trivial de resolver. Las Geociencias[1] se han ocupado de diseñar diferentes técnicas y herramientas para lograr modelar de la manera más exacta posible la superficie de la Tierra, y poder así establecer un sistema de medición de posiciones sobre ella. A esto le

[1]Geodesia en particular, que es la ciencia que se ocupa de fijar la forma de la Tierra.

sigue la necesidad de representar la superficie de un modelo de la tierra en tres dimensiones, en un plano.

Lo primero que se hace para representar la superficie terrestre en un plano, es modelarla como un elipsoide.

Definición 2.1.2 (*Elipsoide*)

Un elipsoide es la superfice definida por la ecuación[2]*:*

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

Los parámetros a, b y c son denominados los semiejes del elipsoide, respecto de x, y y z respectivamente.

La superficie del *geoide* es entonces aproximada con un elipsoide. Dados los semiejes de éste, y una inclinación, un elipsoide logra aproximarse mejor al *geoide* en la medida en que existe la mayor cantidad de puntos coincidentes entre la superficie definida por éste y por el *geoide*.

Definición 2.1.3 (*Datum*)

Dado el geoide $G \subset \mathbb{R}^3$ y un elipsoide definido por

$$E = \{(x, y, z) \in \mathbb{R}^3 : \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1\}$$

se define el conjunto $D = G \cap E$. Un punto $p \in D$ es denominado conjunto de puntos datum.

La determinación del *datum* (y por lo tanto, infiriendo de la Definición 2.1.3, del elipsoide; en la Figura 2.2 se puede apreciar un llamado punto datum) permite establecer una referencia para un sistema de coordenadas con el cual medir la posición en el modelo de la Tierra. Sin embargo, dada la particular forma del *geoide*, los *datum* son sólo de alcance regional, lo mismo que los elipsoides de referencia: dependiendo de la zona geográfica modelarán con mejor o peor precisión la superficie de tal zona.

Elipsoide y *datum* conforman lo que se llama un Sistema de Referencia, a partir del cual se posicionan los objetos en la Tierra fijando el sistema de coordenadas (llamadas coordenadas geodésicas). Un Sistema de Referencia clásico está entonces compuesto por:

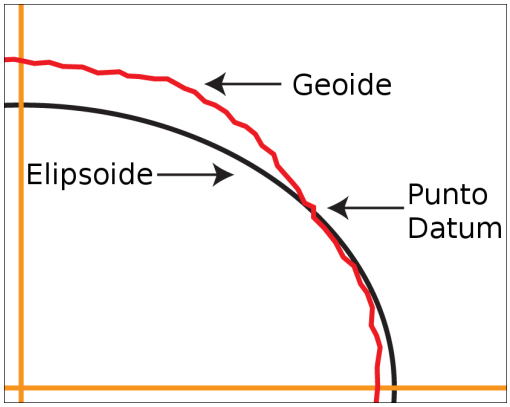

Figura 2.2: Punto datum.

- Un elipsoide de referencia.
- Un datum geodésico[3].
- Un origen de latitudes y de longitudes (que corresponde a la inclinación del elipsoide).

Dentro de los Sistemas de Referencia (o elipsoides de referencia) más utilizadas se encuentran los siguientes: Clarke 1866 y 1880 IGN, Bessel , Airy, Hayford 1909, International 1924 y 1967, WGS 66, WGS 72 y WGS 84, IAG-GRS80, NAD27 y NAD 83.

Uno de los más populares es el World Geodetic System del año 1984 (WGS 84), definido por el Departamento de Defensa de Estados Unidos, y utilizado por la mayoría de los GPS.

En el caso particular de Chile, dos *datum* son comúnmente usados: *Provisional South American Datum 56* (PSAD 56), cuyo punto de origen se encuentra en La Canoa, Venezuela; y *South American Datum 69* (SAD 69), cuyo punto de origen está en Chua, Brasil.

2.1.2. Sistema de proyección.

Una vez definido el elipsoide que modela el *geoide* terrestre, surge la necesidad de representar la superficie de este elipsoide en un plano. Para esto debe definirse un método que mapee los puntos desde una superficie a otra, conservando, en la medida de lo posible, algunas propiedades básicas como distancias, ángulos, áreas, forma de las superficies o direcciones. Para esto se definen las proyecciones.

[3]Un punto fundamental de referencia para el sistema.

Definición 2.1.4 (*Proyección*)

Dado un elipsoide $E \subset \mathbb{R}^3$, una proyección geográfica es una función $f : E \to \mathbb{R}^2$, que mapea los puntos del elipsoide a un plano cartesiano de 2 dimensiones.

Existe una gran cantidad de proyecciones, clasificables según varios criterios (ver Cuadro 2.1).

Criterio	**Tipos**
Método de Construcción	En perspectiva Sin perspectiva
Preservación de propiedades	Homológráfica (preserva áreas) Ortomórfica Conforme (preserva ángulos)
Forma de desplegar área	Cilíndrica Cónica Azimutal
Posición del plano tangente	Polar Normal Oblicua

Cuadro 2.1: Tipos de Proyecciones

La clasificación más común de las proyecciones es la forma en que se proyectan los sectores del elipsoide en un plano. La figura 2.3 ilustra a grandes rasgos la forma en que se realiza la proyección en el plano para cada tipo.

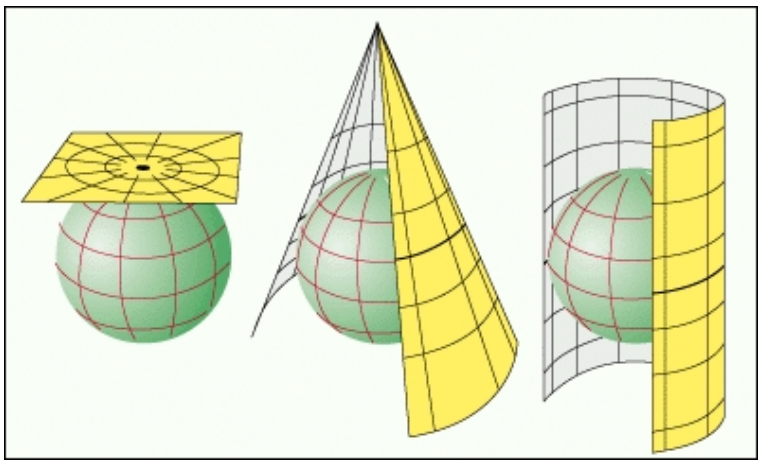

Figura 2.3: Esquema de proyecciones de la tierra: Azimutal, Cónica y Cilíndrica.

Hay que considerar que una proyección será mejor que otra, del mismo modo que los elipsoides, sólo a nivel regional. Por ejemplo, si consideramos una proyección cónica tomada

desde un polo, y la comparamos con una esférica que haga tangencia en la Línea del Ecuador ecuador, podemos decir que:

1. La proyeccón cónica sólo tiene sentido en un hemisferio del elipsoide, mientras que la cilíndrica sirve para mapear el elipsoide completo.
2. La proyección cónica es más precisa que la cilíndrica en el polo.
3. La proyección cilíndrica es más precisa en el ecuador, y va perdiendo precisión a medida que se aleja hacia los polos.

Entre las proyecciones más utilizadas podemos mencionar: Proyección de Mercator, Proyección de Peters, Proyección de Robinson, Proyección Gauss-Krugger, Proyección de Lambert, Proyección de Cassini, Proyección estereográfica, Proyección ortográfica, entre otras (ver Figura 2.4, que ilustra el resultado al proyectar de distintos modos el elipsoide).

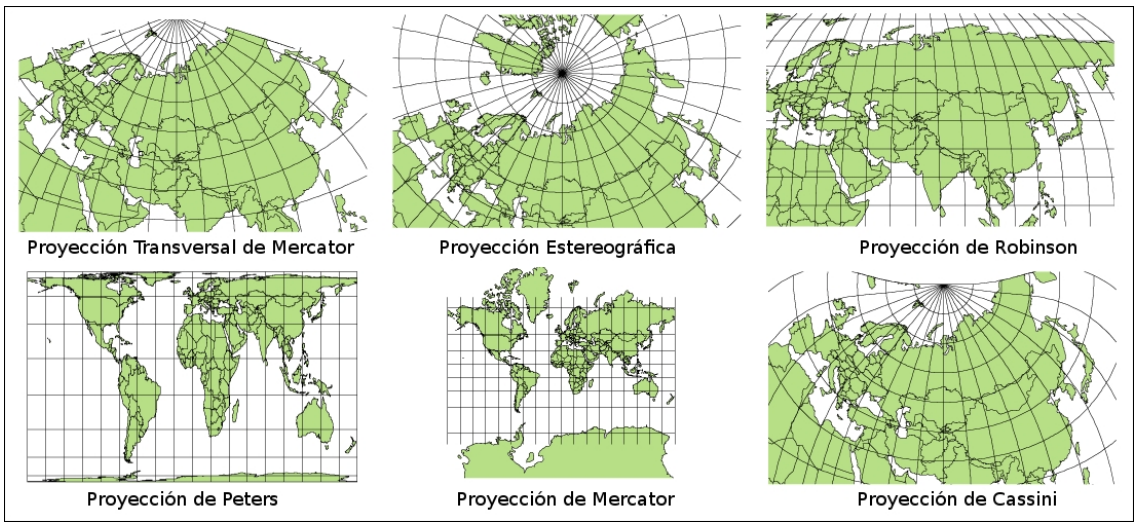

Figura 2.4: Ejemplos de proyecciones geográficas.

La más utilizada dentro de las proyecciones globales (aquellas que mapean el elipsoide completo) es la Proyección de Mercator. Esta es una proyección cilíndrica y conforme (esto es, que conserva los ángulos) formalizada por Gerardus Mercator en el siglo XVI, y base para el sistema de coordenadas UTM[4].

[4]Universal Transverse Mercator.

Coordenadas UTM.

El Sistema de Coordenadas UTM se basa en la proyección de Mercator de un elipsoide terrestre (comúnmente WGS 84), y define un reticulado del globo consistente en 60 zonas longitudinales[5] (cada una de 6 grados sexagesimales) y 20 bandas de latitud (cada una de 8 grados sexagesimales). Un punto en el mapa bajo este sistema de proyección es definido por sus coordenadas Norte y Este, medidas en metros.

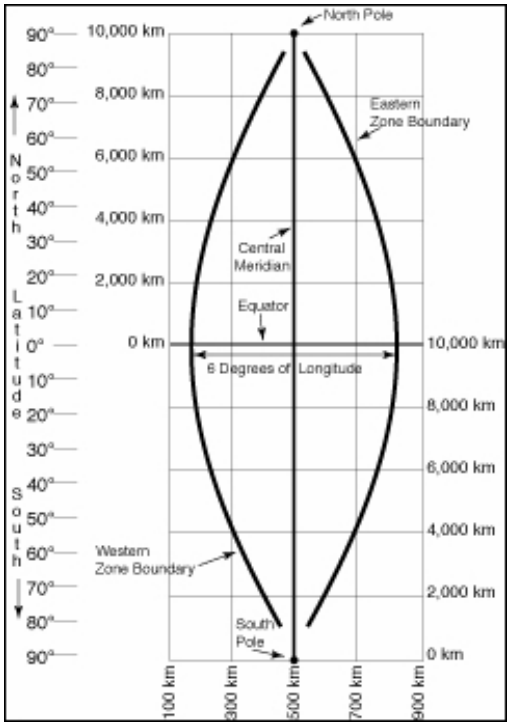

Figura 2.5: Gráfica de una zona UTM.

Cada una de las zonas es separada en el ecuador, donde se marca el origen para la coordenada Norte del sector norte de la zona, y el valor de 10.000.000 metros del sector sur (la Figura 2.5 ilustra cómo se divide y define un sistema de referencia en una zona UTM). En el caso particular de Chile, éste se distribuye entre las zonas 18 y 19 Sur; esta situación hace que los datos geográficos del país se refieran a ambos husos, y en general para puntos al sur de Concepción se considera en huso 18, mientras que al norte el huso 19.

[5]También denominados husos.

2.1.3. Información geográfica.

La información geográfica tiene algunas particularidades que ameritan un tratamiento especial. Este trato especial tiene relación con el formato para almacenar la información (Sección 2.1.3.1), la manera de intercambiarla (Sección 2.1.3.2), la abstracción de los objetos espaciales (Sección 2.1.3.3) y la manera de representarlos (Sección 2.1.3.4).

2.1.3.1. Almacenamiento: Tipos raster y vectorial, Geodatabase.

Existen dos formas básicas de estructurar la información espacial:

- Vectorial: corresponde a la información espacial construida a partir de geometrías básicas como puntos, líneas y polígonos.
- Raster: corresponde a imágenes en diferentes formatos, para las cuales cada pixel tiene información espacial asociada.

En general la información en formato ráster requiere muchos más recursos computacionales que la vectorial, puesto que esta última sólo necesita almacenar las coordenadas de los vértices de las figuras.

Uno de los formatos más conocidos y utilizados por programas CAD y GIS para archivos con información vectorial son los *shapefiles*, propietario de ESRI. Este formato almacena por separado la información espacial (coordenadas de los vectores), de proyección (elipsoide de referencia, datum) y aquella relacionada a las geometrías.

Respecto de las imágenes raster, existen varios formatos, que dependen de la calidad y las características mapeadas: imágenes satelitales, imágenes de vuelos fotogramétricos o LIDAR (Light Detection and Ranging), radar, entre otros.

Otro concepto relacionado con el almacenamiento, esta vez en una base de datos, es el de la *geodatabase*. Una *geodatabase* es una colección de capas geográficas (vectoriales e imágenes raster) almacenadas en una base de datos que puede ser mono o multiusuario.

El término es acuñado por ESRI, y hace referencia a múltiples conceptos relacionados a varios aspectos [9]:

- La *geodatabase* es la estructura nativa de datos de ArcGIS, y es el formato básico de información utilizado para la edición y administración de la información geográfica.
- Almacenamiento físico de la información geográfica, que utiliza ya sea una base de datos o un sistema de archivos.
- Las *geodatabases* incluyen un modelo de almacenamiento y procesamiento de la información y sus atributos relacionados.

En términos generales, una *geodatabase* será entendida como un RDBMS habilitado, ya sea internamente (a través de procedimientos almacenados o funciones) o a través de un *middleware*, para trabajar con datos espaciales. En particular, aplicaremos el término de *geodatabase* a PostgreSQL extendido con PostGIS.

2.1.3.2. Formatos de intercambio.

El intercambio de información en internet es un tema relevante para cualquier tipo de aplicación *web*. El concepto de redifusión *web* (o sindicación *web*), que tuvo su origen en las páginas de noticias y *blogs*, y su necesidad de compartir sus contenidos actualizados, es aplicable también a las aplicaciones que interactúan con información geográfica.

En efecto, la necesidad de compartir información espacial entre aplicaciones en internet, ha generado la implementación y posterior estandarización de algunos lenguajes de intercambio.

Geography Markup Language

GML es una gramática XML concebida para modelar, intercambiar y almacenar información geográfica en aplicaciones GIS.

La gramática de GML se define en un conjunto de documentos esquema (ver [10]), a partir de especificaciones de un modelo abstracto de geografía generadas por el OGC. Tal modelo describe los posibles objetos de la tierra en términos de entidades geográficas llamadas *features*, que son básicamente una lista de:

- Propiedades: describen metadatos de los objetos, tales como nombre, descripciones de los valores, tipo de geometría, etc.
- Geometría: coresponde a tipos simples de objetos geométricos como puntos, líneas, superficies, polígonos, etc.

En un principio GML fue concebido para modelar geometrías en dos dimensiones; sin embargo se han construido también extensiones para modelar datos en 3D.

Un ejemplo de documento GML es el siguiente:

Ejemplo GML

```
<Feature fid="42" featureType="colegio"
 Description="Un colegio cualquiera">
  <Polygon name="extent" srsName="epsg:27354">
    <LineString name="extent" srsName="epsg:27354">
      <CData>
        491888.9459,5458045.9996 491904.9458,5458044.9996
        491908.9462,5458064.9996 491924.9461,5458064.9996
        491925.9462,5458079.9996 491977.9466,5458120.9996
        491953.9466,5458017.9996
      </CData>
    </LineString>
  </Polygon>
</Feature>
```

En este ejemplo se describe una *feature* que representa un colegio, descrito geométricamente como un polígono, a su vez compuesto por un conjunto de líneas. Otra característica a notar es el hecho de que un documento GML incluye también información acerca del contexto de los objetos, como el tipo de proyección geográfica (en este caso el atributo `srsName` de los nodos `Polygon` y `LineString`).

Al igual que en XML, es posible dar un formato visual a un documento GML a partir de documentos de estilo (XSLT).

GML es primordialmente utilizado para el intercambio de información espacial entre las aplicaciones GIS (en [11] se describen técnicas para modelar objetos en movimiento con GML). Sin embargo, una de las principales desventajas que plantea este formato es el tamaño de los archivos[6]; en [12] se proponen algunas técnicas para la optimización en el trabajo con este tipo de archivos, tales como el procesador de consultas en GML, GPXQuery, y el parser GPSAX.

[6]Sucede algo similar con documentos KML, que suelen comprimirse (KMZ).

Keyhole Markup Language

KML es un subconjunto de XML definido por Google (y estándar de la OGC, especificado en [13]) enfocado principalmente en la visualización de información geográfica, incluyendo anotaciones en mapas vectoriales o raster. Es utilizado principalmente en visualizadores de mapas como Google Earth, Google Maps y Google Maps Mobile.

La sintaxis de KML es muy similar a GML, utilizando los elementos de este último para describir tipos geométricos básicos como puntos, líneas y polígonos. El acuerdo entre Google y OGC es homologar ambos lenguajes en el futuro.

El siguiente es un ejemplo de una línea descrita en KML [14]:

Ejemplo KML

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

De este ejemplo es posible apreciar que, junto con la información espacial de los vectores, KML agrega información asociada como anotaciones. El elemento `Placemark` representa una de las funcionalidades más utilizadas por Google Earth, y que corresponde a etiquetas de información georreferenciada. En el ejemplo particular está compuesto por un nombre y una descripción, asociados a un punto en coordenadas geográficas (angulares, o *latlon*).

GeoRSS

La necesidad de contar con mecanismos estándar para compartir información en la *web* se extiende para el caso especial de la información geográfica. Para este propósito se ha definido un esquema de interoperabilidad que permite compartir fuentes RSS de mapas: GeoRSS.

En la actualidad existen dos tipos de codificación de fuentes GeoRSS: Simple y GML. La primera es una versión más ligera de GML que incluye los elementos básicos (las geometrías básicas, por ejemplo) para compartir ubicaciones; la segunda, una aplicación de GML, ofrece

más funcionalidades, incluyendo el manejo de información en diferentes sistemas de coordenadas.

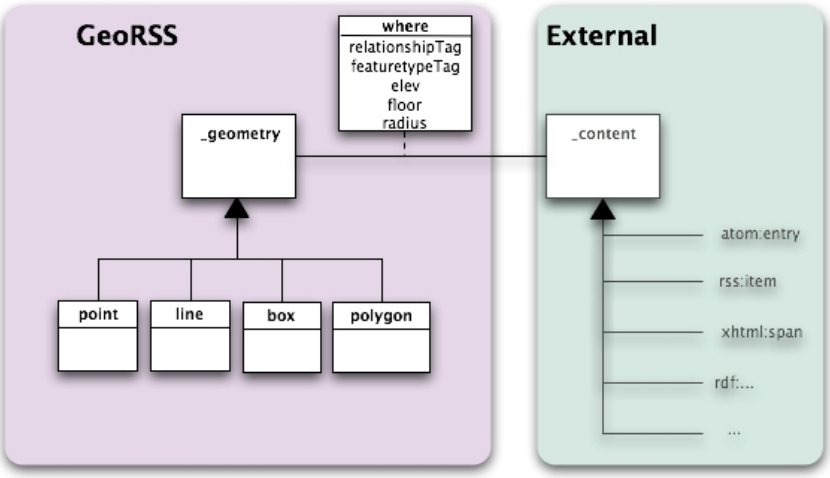

Figura 2.6: Modelo de GeoRSS.

En la Figura 2.6 se puede aprecia el modelo para GeoRSS. El bloque de la derecha representa GeoRSS, y el de la izquierda una fuente de contenido a la cual se le agrega información espacial a través del elemento `where`. Esta asociación tiene sentido en un documento XML Atom o RSS concreto[7].

Se puede apreciar también en la Figura 2.6 el uso de cuatro tipos geométricos básicos para describir ubicación espacial: Punto, Línea, Caja (Box, también llamado Sobre) y Polígono.

2.1.3.3. *Simple Features.*

Simple Features es un estándar OpenGIS para la representación de los objetos del mundo real, y el almacenamiento de esta representación (ver [15] y [16]). Las entidades del mundo real son representadas utilizando tipos geométricos básicos en 2D (con iterpolación lineal entre vértices): Punto, Línea, Polígono.

Definición 2.1.5 (*Punto*)

Un punto es una geometría adimensional que representa una ubicación en un espacio coordenado. Es un elemento de $\mathbb{R}^2$.

[7] Actualmente se encuentra bajo desarrollo para RDF y XHTML.

Antes de definir una línea[8], se define una curva:

Definición 2.1.6 (*Curva*)

Sea $D = \{x \in \mathbb{R} : a \leq x \leq b\}$, *y un homomorfismo* $f : D \to \mathbb{R}^2$. *Una curva es la imagen de* f.

Una curva se dice simple ssi:

$$(\forall x_1, x_2 \in (a, b], x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)) \wedge (\forall x_1, x_2 \in [a, b), x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2))$$

Una curva se dice cerrada ssi $f(a) = f(b)$.

Una curva simple y cerrada se llama anillo.

Definición 2.1.7 (*Línea*)

Una línea es una curva con interpolación lineal entre algunos puntos.

Del mismo modo que las curvas, las líneas pueden ser simples y/o cerradas (ver Figura 2.7, con ejemplos de las posibles combinaciones de líneas).

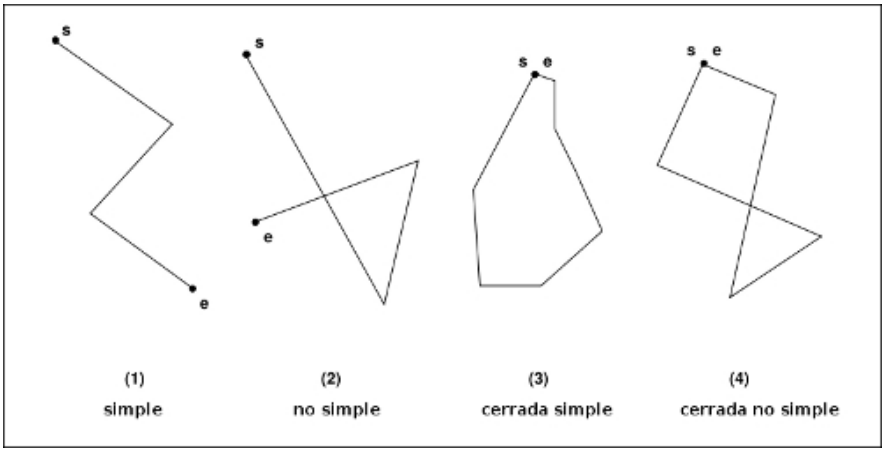

Figura 2.7: Ejemplos de líneas.

Análogamente a las líneas, antes de definir un polígono se define el tipo superficie:

Definición 2.1.8 (*Superficie*)

Una superficie (simple) es un área a la cual se le asocian tanto un único borde exterior como cero o más bordes interiores, entendiendo por bordes líneas simples y cerradas (anillos).

[8]O *line string*; en la especificación de las *Simple Features* se hace la diferencia entre una *linestring* (línea quebrada conformada por varios puntos) y una *line* (línea recta). En este documento nos referimos a la primera como línea.

Definición 2.1.9 (*Polígono*)

Un polígono es una superficie plana definida por un borde exterior o cero o más bordes interiores. Cada borde interior define un agujero en el polígono.

Para efectos de preservar la correctitud de las geometrías y la consistencia en su operatoria, se establecen ciertas reglas que definen un polígono válido:

Definición 2.1.10 (*Polígono válido*)

Las siguientes reglas definen un polígono válido:

- *Un polígono es topológicamente cerrado (contiene sus bordes).*
- *El borde de un polígono está compuesto por un conjunto de líneas quebradas cerradas que constituyen sus bordes interior y exterior.*
- *Dos anillos correspondientes a los bordes de un polígono no se intersectan sino tangencialmente.*
- *El interior de un polígono es conexo.*
- *El exterior de un polígono con uno o más agujeros es inconexo.*

La Figura 2.8 ilustra ejemplos de polígonos que pueden ser representados bajo las reglas recién descritas (polígonos 1 al 3), y de polígonos que violan estas reglas (4 al 7; los polígonos 4 y 7 pueden ser representados a partir de dos polígonos individuales).

2.1.3.4. Representaciones WKT y WKB.

En las especificaciones para manejo de geometrías, el OGC propone dos maneras estándar para manejar objetos espaciales [16]: *Well-Known Binary* (WKB) y *Well-Known Text* (WKT). Ambas incluyen información acerca del tipo de geometría y las coordenadas que definen al objeto.

Well-Known Text

WKT es utilizada para dar una representación alfanumérica de cualquier geometría. Esta representación debe ser conforme a una gramática (definida también en [16]). Ejemplos de geometrías en WKT son los siguientes:

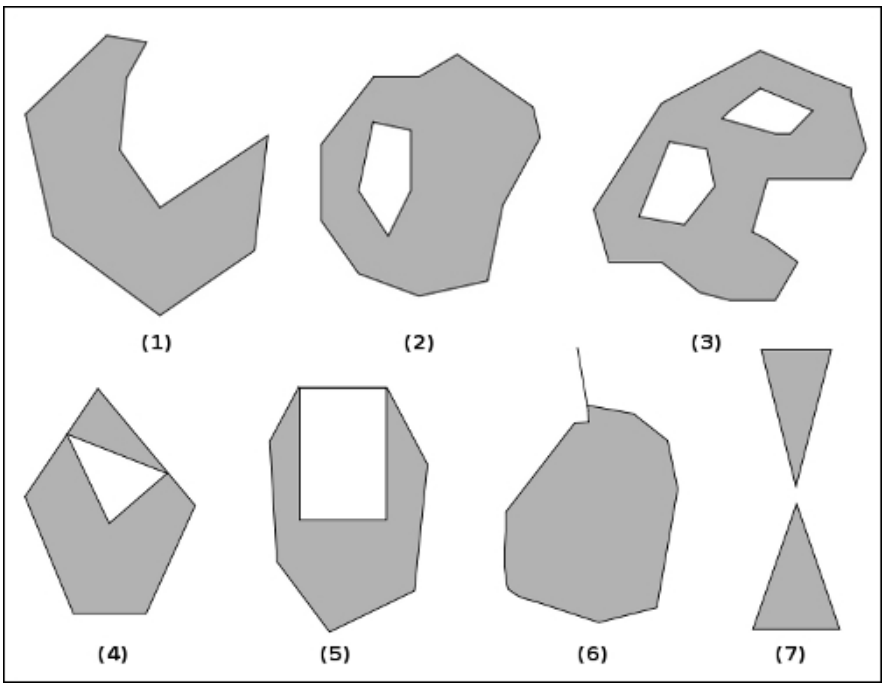

Figura 2.8: Ejemplos de polígonos válidos y no válidos.

Ejemplos WKT

```
POINT(0 0)
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
```

Suelen definirse también extensiones de la gramática que soportan la representación de objetos en 3D, la dimensión adicional *measure*, y la inclusión de atributos como el Sistema de Referencia. Tales extensiones incluyen al estándar definido por la OGC.

Well-Known Binary

La representación en WKB de una geometría se construye con combinaciones de números (enteros sin signo de 4 bytes y reales de doble precisión de 8 bytes) en algún formato binario de representación (NDR o XDR[9]), ordenados de acuerdo a un formato establecido en [16].

La Figura 2.9 muestra un ejemplo de un polígono en WKB[10]. Cada bloque de bytes repre-

[9]NDR (Native Data Representation) corresponde a la representación nativa del computador específico; XDR (External Data Representation) es una representación independiente de la máquina que facilita el transporte de información. Ambos difieren, en las representaciones que interesan en este caso (enteros sin signo y dobles), en el orden de representación: NDR representa en *Little Endian* y XDR en *Big Endian*.

[10]La construcción de cualquier tipo de geometría se puede hacer siguiendo la especificación descrita en [16]; en términos simples, una geometría en WKB se forma con un grupo de bloques de bytes de encabezado, seguidos por bloques de bytes que representan los valores de las coordenadas de los elementos que definen la geometría.

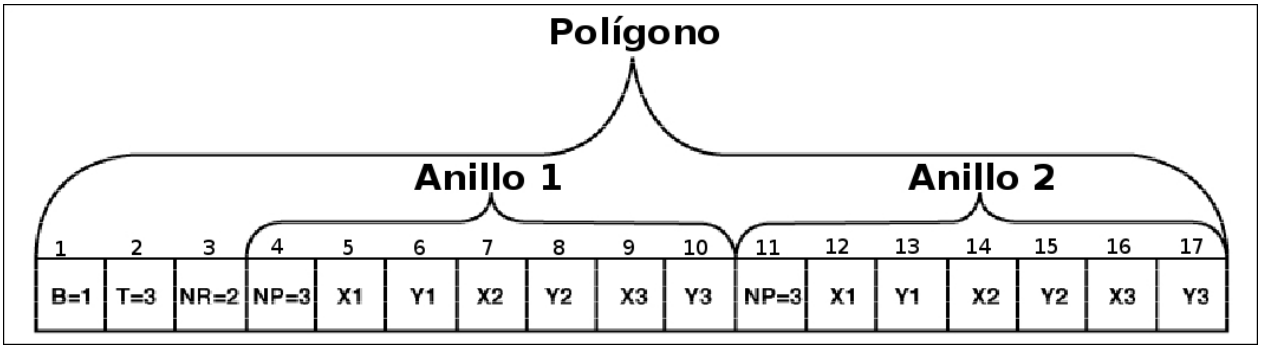

Figura 2.9: Ejemplo de un polígono en WKB.

senta lo siguiente:

- El bloque 1, de 4 bytes, indica el tipo de representación (`B=1` para NDR).
- El bloque 2, de 4 bytes, indica el tipo de geometría representada (`T=3` para polígono).
- El bloque 3, de 4 bytes, indica la cantidad de anillos que definen el polígono (en este caso 3).
- Los bloques 4 al 10, y 11 al 17, se usan para representar los anillos. El primer bloque de la secuencia indica la cantidad de puntos que compone el anillo (4 bytes), descritos luego mediante sus coordenadas (8 bytes).

2.2. Software GIS: Estado del arte.

2.2.1. Herramientas propietarias.

Desde el primer desarrollo considerado como un GIS en los años '60 [17], el mercado de las aplicaciones para la visualización gráfica de la información geoespacial ha evolucionado rápidamente. Al día de hoy existe una gran oferta de aplicaciones destinadas a diferentes propósitos.

De los proveedores más importantes existentes en el mercado, se destaca ESRI, que entrega un completo conjunto de aplicaciones de escritorio y servidoras para ayudar en las diferentes etapas del manejo de la información geoespacial, desde la captura hasta su publicación, pasando por el procesamiento, análisis, modelamiento y almacenamiento.

La forma en la cual se instrumentan este grupo de herramientas para sostener un GIS corporativo, se puede apreciar en la Figura 2.10. En términos generales, un servidor GIS es el encargado de proporcionar acceso a una *geodatabase* a los diversos clientes, tanto en ambiente *web* como de escritorio.

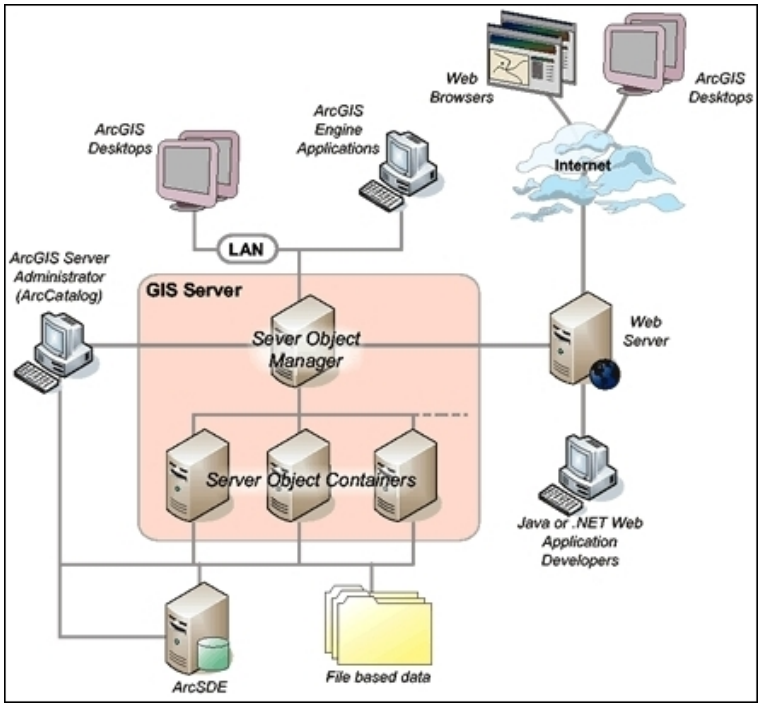

Figura 2.10: Despliegue típico de software ArcGIS.

La principal ventaja de montar una infraestructura de *software* GIS usando ArcGIS, es la simplificación de los procedimientos para el manejo de la información geográfica. En el particular interés de este trabajo, se destaca la facilidad con que es posible publicar, y luego extender, una Aplicación Web de Mapas a partir de los datos disponibles en la *geodatabase*. Incluso para usuarios sin conocimientos de programación, existen herramientas como ArcGIS Manager, que permite la creación y personalización de las Aplicaciones Web de Mapas a partir de simples pasos.

Ejemplos de implementaciones de aplicaciones Web GIS con herramientas de ESRI se detallan en [18] (modelo de cursos de aguas) y [19] (experiencia de integración de plataformas para levantar un GIS). En [20] se describe un caso de uso para la publicación de un servicio web de mapas de gran capacidad, a través de ArcGIS Server, de los datos del programa Europeo de Coordinación de Información del Medioambiente (CORINE). El conjunto de datos publicados consta de aproximadamente 2 millones de objetos espaciales, algunos de gran complejidad (60 mil vértices), que encierran información de uso de suelo, fuentes de agua, calidad del aire, entre otra información medioambiental. Estos datos son publicados para usuarios generales (que acceden a los datos visitando las Aplicaciones Web de Mapas a través de navegadores *web*) y expertos (que acceden a través de clientes GIS como ArcGIS Desktop o AutoCAD), bajo requerimientos como:

- Alto desempeño para responder a las solicitudes de mapas en menos de un segundo.
- Escalabilidad: debe soportar cientos de usuarios concurrentes sin degradar el servicio.
- Disponibilidad de los datos.
- Posibilidad de soportar diferentes proyecciones cartográficas según preferencias de los usuarios (Lambert, Mercator, Estereográfica, etc.).
- Posibilidad de desplegar los mapas en múltiples escalas (1:25000 para análisis local, 1:1000000 para análisis nacional).
- Manejo simple de la información desplegada, desplegando y ocultando capas de información, y realizando consultas espaciales o sobre los atributos alfanuméricos.
- Interoperable según estándares como el del OGC.

Se describen a continuación cada una de las aplicaciones de ESRI, enfatizando el propósito específico para el cual están construidas.

2.2.1.1. ArcGIS Desktop.

ArcGIS Desktop es una *suite* de escritorio compuesta por tres paquetes de aplicaciones: ArcInfo, ArcEditor y ArcView. Estos paquetes, que difieren uno del otro básicamente en la cantidad de funcionalidades que ofrecen[11] (como se grafica en la Figura 2.11), están destinados principalmente a los productores de la información. Esto es, profesionales del área de la Geografía o Geomática encargados de levantar y administrar la información geoespacial de una organización, además de distribuirla a las áreas que la requieran.

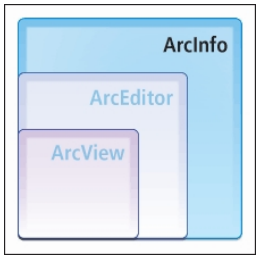

Figura 2.11: Inclusión de funcionalidades en ArcGIS Desktop.

Las funcionalidades que ofrece cada paquete de ArcGIS Desktop son:

- ArcInfo: es la más completa de las herramientas de ArcGIS Desktop. Permite administrar la información geográfica durante todo el ciclo de vida de ésta, ofreciendo la posibilidad de crear y administrar geodatabases multiusuario. El gran fuerte de ArcInfo está en un conjunto de herramientas (el *toolbox*), que además de disponer un gran número de procesos encapsulados y listos para utilizar[12], es fácilmente extensible gracias a una interfaz gráfica para la construcción de nuevas herramientas por parte del usuario.

- ArcEditor: corresponde al nivel de licenciamiento siguiente a ArcInfo. Este cliente de escritorio permite básicamente editar la información geográfica disponible en variados formatos, como archivos *shapefiles* o geodatabases. Es utilizado principalmente para la

[11] Y por lo tanto, en la forma y costo del licenciamiento.
[12] Análisis en 3D, procesamientos geoestadísticos, corrección de geometrías, etc.

producción y estandarización de las capas geográficas que conforman el repositorio de datos geoespaciales de una organización.

- ArcView: es la más simple de las licencias ofrecidas en el paquete. Está destinada principalmente a la visualización de la información geográfica desde distintas fuentes, tales como archivos shapefiles locales o en red, gepdatabases multiusuario y WMS.

2.2.1.2. ArcGIS Server.

ArcGIS Server es un conjunto de tecnologías cuyo propósito principal es simplificar a los usuarios de una organización, ya posean éstos o no conocimientos en GIS, el acceso a la información geográfica disponible. Con ArcGIS Server es posible controlar el contenido a través de la gestión centralizada de los datos en una *geodatabase*, incluyendo capas vectoriales e imágenes raster, pudiendo ser desplegados en distintos soportes (ver Figura 2.12, que ilustra el flujo de información manejado por ArcGIS Server).

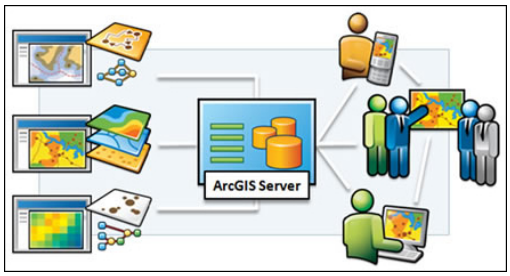

Figura 2.12: Publicación de la información con ArcGIS Server.

La plataforma de ArcGIS Server es escalable, pudiendo desplegarse tanto en una única máquina o en un conjunto de servidores para prestar servicios a diversos departamentos en una organización[13].

Las funcionalidades centrales de ArcGIS Server son las siguientes [21]:

- Gestión de *geodatabase*: corresponde al manejo de la información disponible en una *geodatabase*.

[13]Por ejemplo, en las compañías mineras suele existir la figura de un Administrador GIS que sirve a departamentos que producen y consumen información geográfica, como Propiedad Minera o Gestión Territorial.

- Réplica: corresponde a la replicación asincrónica de la *geodatabase*, de tal modo que los usuarios editores tengan la capacidad de modificar la información mientras está publicada[14].

- Servicios de Mapas y Aplicaciones Web de Mapas: corresponde a la pulicación de la información geográfica, ya sea como servicios o aplicaciones *web*.

- Edición de capas: funcionalidad que permite la edición de las capas desde las Aplicaciones Web de Mapas.

- Geoprocesamiento: corresponde a tareas básicas de geoprocesamiento, tales como medición de áreas y distancias e impresión a escala.

- Geoprocesamiento avanzado: corresponde a un conjunto de herramientas de geoprocesamiento como las incluídas con ArcInfo, disponibles con ArcGIS Server para las aplicaciones *web*.

- SDK para desarrollo: corresponde a un completo *kit* de desarrollo para aplicaciones *web*.

Sumadas a estas funcionalidades, existen además un conjunto de extensiones que agregan funcionalidades para visualización de datos en 3D, análisis de redes, creación de flujos de trabajo con información geográfica, geoestadística, etc.

El espectro de usuarios de la plataforma de ArcGIS Server incluye tanto a quienes tienen experiencia GIS como a quienes no [22]:

- Profesionales GIS: utilizan ArcGIS Server para publicar y promover su trabajo, que incluye mapas, imágenes raster, tareas de geoprocesamiento y flujos de trabajo de información.

- Desarrolladores: para quienes desarrollan, ArcGIS Server entrega un conjunto de tecnologías (APIs, *frameworks*, controles *web*) para diferentes lenguajes (.NET, Java,

[14] Al publicar una capa de información geográfica a través de un Servicio en ArcGIS Server, ésta no puede ser editada en la *geodatabase*. Por esta razón, se crea una réplica de la *geodatabase* para la publicación.

Adobe Flex, Javascript y Silverlight) que apoyan el desarrollo de aplicaciones de dominio específico que involucren manejo de información geográfica.

- Trabajadores en terreno: pueden tanto acceder a la información geográfica en la *geodatabase*, como levantar nueva información, editar y realizar análisis espacial en línea.
- Gerentes de TI: son los encargados de integrar el GIS con otros servicios en la organización, como gestión documental.
- DBA: Son los encargados de mejorar el manejo de la información geográfica reduciendo la duplicidad e incrementando el desempeño.

ArcGIS Server es comercializado en tres tipos de licencia, cada una agrupando distintas funcionalidades (tanto centrales como extensiones): Básica, Estándar y Avanzada.

2.2.1.3. ArcSDE.

Tecnología que en las últimas versiones viene incluida dentro de la plataforma ArcGIS Server. El principal objetivo de ArcSDE es extender los Sistemas de gestión de Bases de Datos comunes como Oracle, MS SQL Server o PostgreSQL, para que soporten datos espaciales, funcionando como un *middleware* entre las aplicaciones GIS y la base de datos. ArcSDE ofrece, además, funcionalidades para el control de versiones de las geodatabase, muy útil en entornos de edición multiusuario.

2.2.1.4. Web Application Developer Framework.

El Web ADF de ArcGIS, disponible para .NET y Java, es un conjunto de controles *web*, clases, *frameworks* y APIs que permiten a los desarrolladores crear aplicaciones en un entorno *web* agregándole características GIS, o extender a un dominio específico una plantilla de Aplicación Web de Mapas incluída con ArcGIS Server.

El *kit* completo[15] consiste en cuatro grupos de herramientas:

[15]Para el caso particular de .NET.

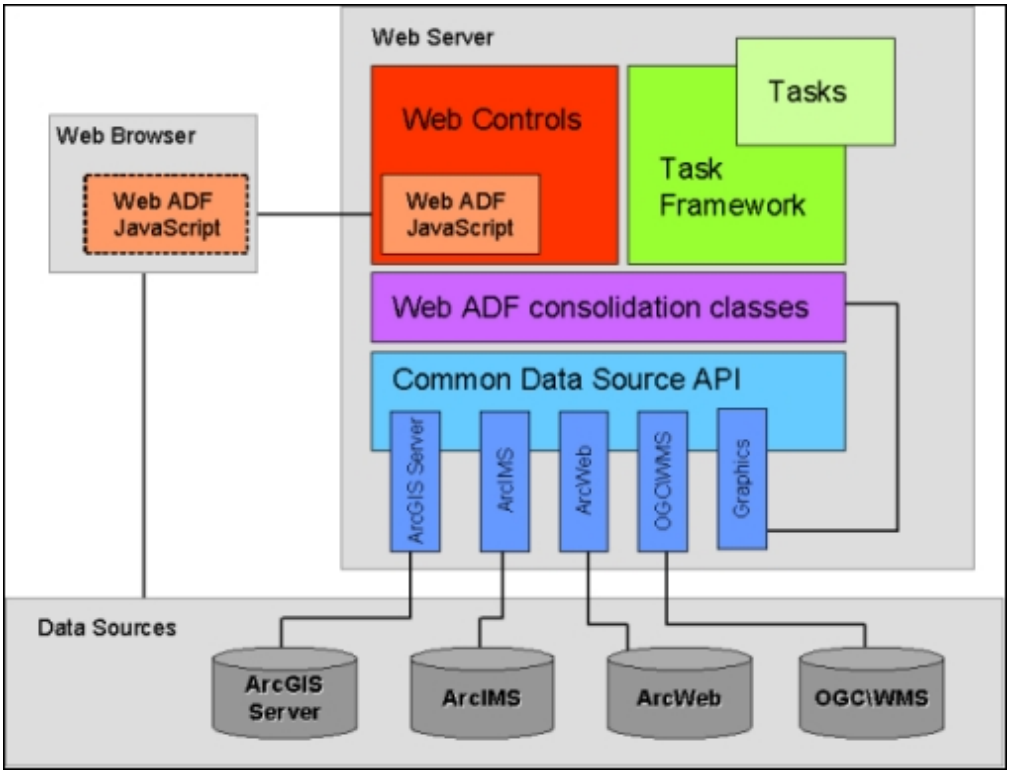

Figura 2.13: Componentes del Web ADF.

- Controles Web: el Web ADF incluye un conjunto de controles *web* AJAX que extienden la arquitectura de ASP.NET. Estos controles proporcionan lo necesario para procesar interacciones de modo asíncrono entre el navegador y la aplicación *web* de mapas, así como con recursos remotos (como pueden ser repositorios de datos).

- *Framework* para tareas web: el Web ADF proporciona mecanismos para encapsular tareas *web* en controles personalizados.

- API de acceso a datos: consiste en un conjunto de clases e interfaces agrupados en una librería. Estas clases e interfaces conforman el marco común mediante el cual los controles *web* acceden a los repositorios de datos geográficos.

- Clases gráficas y de consolidación: corresponde a un conjunto de componentes diseñado para dar soporte al desarrollo en las diferentes capas de una aplicación *web*.

La Figura 2.13 ilustra cómo se agrupan e interactúan estos componentes.

2.2.2. Herramientas libres.

Dentro de las herramientas de código abierto disponibles para el desarrollo de aplicaciones GIS, y Web GIS en particular, no podemos encontrar una análoga a los productos ofrecidos por ESRI con ArcGIS, en lo que a completitud se refiere. Lo que hay son componentes individuales que, integrados de una buena manera, hacen posible obtener una plataforma de *software* suficiente para sostener un GIS.

Si correspondiese hacer la analogía entre la plataforma propietaria de ESRI y las tecnologías de código abierto, podría resumirse en el Cuadro 2.2.

Herramienta ESRI	**Herramienta código abierto**
ArcGIS Desktop	Quantum GIS
ArcGIS Server	MapServer
Web ADF	PHP MapScript
ArcSDE	PostGIS

Cuadro 2.2: Analogía entre tecnologías ESRI y de código abierto.

2.2.2.1. PostGIS.

Desarrollado por Refractions Research Inc., PostGIS es una extensión del RDBMS PostgreSQL que permite el almacenamiento de objetos GIS en una base de datos, habilitándolo así como repositorio de datos espaciales para cualquier sistema GIS[16]. Haciendo una analogía, PostGIS cumple el mismo rol que ArcSDE dentro de la tecnología ArcGIS Server.

PostGIS está implementado conforme a las especificaciones del OGC descritas en [15], y en palabras simples, podemos decir que se trata de:

- Dos tablas, una para el manejo de los sistemas de referencia, y otra para mantener un catálogo de las tablas en la base de datos que contienen una columna de tipo geometría.
- 685 funciones (o procedimientos almacenados) que implementan la completa gama de operaciones espaciales entre los objetos GIS, para ser usadas en consultas SQL.

[16]http://postgis.refractions.net/

2.2.2.2. MapServer.

MapServer es una plataforma de código abierto para la publicación de información geoespacial y el desarrollo de aplicaciones *web* de mapas interactivas. Es multiplataforma (Windows, Linux y MacOS), y soporta una gran lista de entornos de desarrollo[17]. Permite además desplegar datos de distintos formatos de archivos raster, vectoriales y bases de datos.

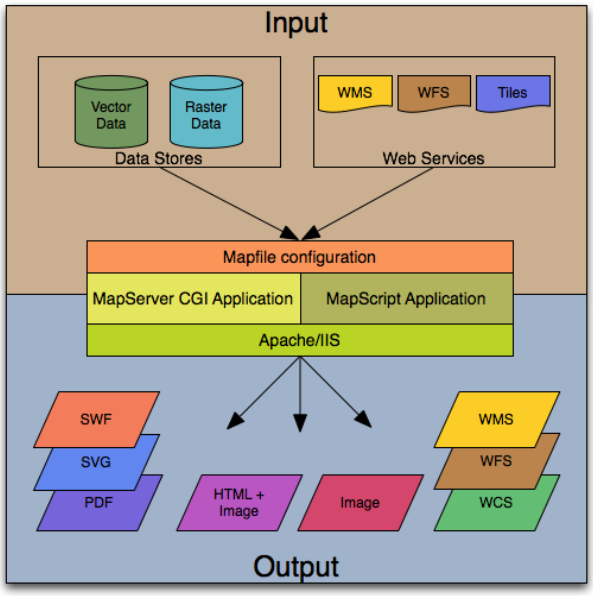

Figura 2.14: Arquitectura MapServer

En términos generales, tal como puede apreciarse en la Figura 2.14, MapServer opera como una aplicación CGI que atiende peticiones. A partir de algunos parámetros enviados en una petición, y de un archivo *mapfile* donde se especifican, entre otras cosas, las fuentes de datos, MapServer genera y entrega como respuesta un mapa en distintos formatos: imagen, WMS, SVG, entre otros.

2.2.2.3. Quantum GIS.

Quantum GIS (QGIS) es una aplicación GIS de escritorio multiplataforma (Windows, Linux y MacOS), cuyo propósito general es la visualización y edición de información geográfica en formatos vectorial o raster. QGIS es una interfaz gráfica del GIS GRASS que ha ganado

[17] A través de MapScript, que expone una interfaz para MapServer en distintos lenguajes de programación: PHP, .NET, Java, Ruby, Python y Perl.

popularidad debido al entorno amigable que presenta al usuario. Dentro de la plataforma de *software* que sustenta un GIS organizacional, QGIS es la herramienta con la cual se administra la información geográfica y se producen las salidas como los mapas o planos.

Detalles de diversas implementaciones utilizando este conjunto de tecnologías se pueden encontrar en [23] (despliegue de información espacial relacionada a interés), [24] (WebSAS), [25] (Sahana: herramienta Web GIS para reconstrucción ante desastres naturales) y [26] (generación de mapas a partir de información histórica).

Mención especial merecen las herramientas para la generación de aplicaciones *web* para el despliegue de mapas, dentro de las cuales Chameleon y MapFish son las más populares. Los llamados *frameworks* de generación de aplicaciones *web* de mapas, tienen como propósito principal transparentar, a los usuarios productores de la información, la tarea de publicar información georreferenciada a través de internet. Sin embargo las herramientas existentes tienen dos deficiencias, en el juicio de este trabajo, fundamentales:

- Instalación y configuración compleja. Los paquetes en general requieren de una larga lista de librerías instaladas y preconfiguradas para su funcionamiento, dificultando su utilización.
- Ausencia de un marco de extensión simple para desarrolladores, lo cual limita la creación, a partir de las plantillas, de aplicaciones de propósitos más específicos.

2.3. CodeIgniter.

2.3.1. Descripción del *framework*.

En el ámbito del desarrollo de aplicaciones *web*, durante los últimos años, han adquirido popularidad los llamados *frameworks* para el desarrollo ágil: herramientas que permiten a los desarrolladores implementar, con un mínimo esfuerzo, aplicaciones *web* de propósito general en poco tiempo. Entre muchos otros, podemos mencionar: CakePHP y CodeIgniter para PHP, Ruby on Rails para Ruby, Django y Pylons para Python, MVC.NET para .NET, JSF para Java.

La razón fundamental por la cual se escogió PHP como lenguaje para el desarrollo está relacionada con el mercado: PHP goza de gran popularidad, y es incluido como lenguaje de desarrollo en la mayoría de las plataformas *web* ofrecidas por los proveedores de *web hosting*[18].

Como *framework* base para las extensiones se escogió CodeIgniter, debido a sus características, entre las que se cuentan [27]:

- Permite el desarrollo de aplicaciones dejando una baja huella[19].
- Buen rendimiento.
- Compatibilidad con las tecnologías estándar ofrecidas por la mayoría de las empresas de *web hosting*.
- Requiere configuraciones mínimas.
- No requiere el uso de líneas de comando para la programación (como Django o CakePHP).
- Es flexible, no requiere seguir reglas rígidas al codificar, como aquellas que exigen llamar del mismo modo tablas y clases del modelo, o la existencia de ciertas columnas en la base de datos, etc.

[18]Según estadísticas de NetCraft (`http://news.netcraft.com/archives/2007/08/06/august_2007_web_server_survey.html`) alrededor de un 48 % de los sitios en internet ejecutan bajo Apache. Por otro lado, según la misma fuente, PHP es el módulo de Apache más popular.

[19]Esta característica hace referencia al nivel de personalización que es posible lograr en las aplicaciones desarrolladas con el *framework*.

- No es obligatorio el uso de un lenguaje distinto para las vistas. Con CodeIgniter las vistas se programan en HTML incrustando código PHP.
- Favorece la documentación clara.

Como la mayoría de los *frameworks* mencionados, CodeIgniter implementa el patrón arquitectural Modelo-Vista-Controlador (o MVC), que separa los datos de la aplicación, las interfaces de usuario y la lógica de control en tres componentes distintos (la Figura 2.15 muestra la interacción entre estos componentes).

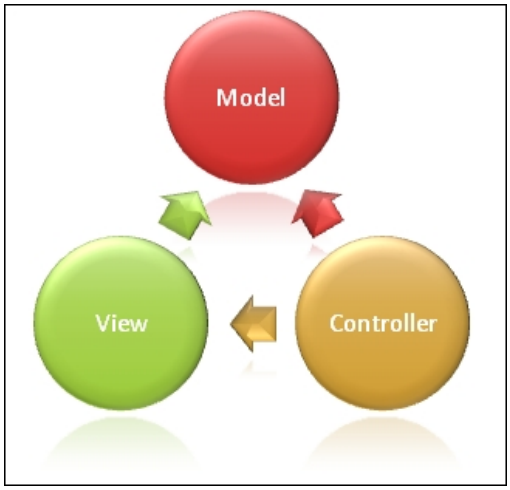

Figura 2.15: Diagrama patrón MVC.

- Modelo: componente encargado de administrar la información, comúnmente se trata de un módulo que implementa el acceso a un repositorio de datos (métodos CRUD).
- Vista: componente encargado de desplegar al usuario la información obtenida desde el modelo. También se encarga de recibir las acciones del usuario.
- Controlador: componente encargado de tomar las solicitudes del usuario y, manipulando modelo y vistas, desplegar la respuesta.

El flujo particular de la información en una aplicación desarrollada con CodeIgniter puede apreciarse en la Figura 2.16. Las solicitudes realizadas desde la interfaz de usuario (que comienzan siempre en un index.php en la raíz del directorio de aplicaciones del *framework*) son primero procesadas por componentes de enrutamiento del *framework*, que a partir de

las URLs pueden definir comportamiento especiales. Luego, si corresponde, se responde con páginas en un *caché* administrado por otro módulo del *framework*. Alternativo a este flujo, cuando no hay respuestas preparadas, la solicitud pasa por filtros de seguridad antes de llegar a la clase controladora. Esta clase es la encargada de acceder tanto al modelo como a las variadas librerías que provee el *framework* para preparar la respuesta en forma de una vista, que aprovecha de ser almacenada en *caché* para futuras solicitudes de la misma información.

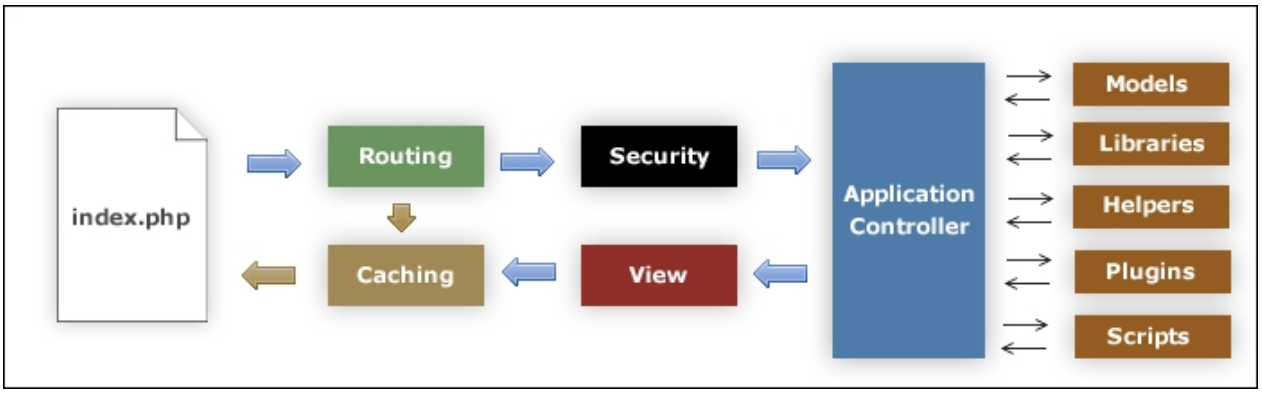

Figura 2.16: Flujo de una aplicación con CodeIgniter.

2.3.2. ¿Cómo extenderlo?.

La estructura modular de CodeIgniter permite tanto la creación como el remplazo o extensión de las clases y librerías nativas del *framework*. Hay, entonces, dos mecanismos para agregar funcionalidades al *framework*:

- Librerías: CodeIgniter provee un mecanismo que posibilita la creación, extensión o remplazo de librerías de utilidades, simplemente creando las clases respectivas en un directorio destinado a ello. Estas librerías son luego cargadas desde los controladores o las clases del modelo en la medida en que son requeridas.
- Clases: CodeIgniter permite la creación, extensión o remplazo de las clases que forman el núcleo del *framework*. Tales clases son inicializadas automáticamente, como la base de CodeIgniter.

Por ejemplo, si requerimos una librería con métodos para convertir temperaturas entre grados Celsius y Fahrenheit, bastará que ubiquemos en el directorio de librerías el siguiente

archivo:

Conversor.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');
class Conversor {
        function Conversor(){}

        function celsius_to_fahrenheit($c)
        {
                return 9/5 * $c + 32;
        }

        function fahrenheit_to_celcius($f)
        {
                return 5/9 * ($f - 32);
        }
}
?>
```

Luego es posible cargar, y utilizar las funciones de la librería nueva (dentro de una clase del modelo o de un controlador) del siguiente modo:

```
        ...
        $this->load->library('Conversor'):
        ...
        $celsius = $this->Conversor->fahrenheit_to_celsius(100);
        ...
```

Capítulo 3

Desarrollo.

En este capítulo se describen los detalles del desarrollo realizado. Este desarrollo ha sido separado en dos partes: la primera, comprende la implementación de las librerías para el manejo de datos geoespaciales; la segunda, es el desarrollo de la aplicación generadora de Aplicaciones Web de Mapas. Consecuentemente, el capítulo se divide en dos secciones, cada una dando cuenta del desarrollo correspondiente.

Respecto de las librerías que extienden el *framework*, éstas fueron implementadas conforme a estándares definidos por el OGC (ver [15]), y enfocadas a satisfacer las necesidades planteadas al comienzo del desarrollo, a saber: permitir extender las Aplicaciones Web de Mapas, creadas con una aplicación *web* administradora (correspondiente a la segunta etapa de esta implementación).

Respecto de la aplicación administradora, la cual puede considerarse como el producto final y más visible de este trabajo, fue implementada con la vista en la aplicación ArcGIS Manager, incluida en el paquete de ArcGIS Server. Pretende igualar sus funcionalidades, y permitir así que los responsables (productores y/o administradores) de la información geográfica la compartan al resto de sus organizaciones, de un modo simple.

3.1. Librerías geográficas para CodeIgniter.

Esta parte del desarrollo tenía como propósito equipar al *framework* de desarrollo escogido con lo necesario para el desarrollo de aplicaciones Web GIS, y en particular para la extensión de las Aplicaciones Web de Mapas creadas con la herramienta administradora. Para esto, se implementó un modelo de geometrías para trabajar con los objetos espaciales (*Simple Features*), además de diversas librerías para el trabajo con archivos vectoriales (*shapefiles*), intercambio de información (GeoRSS, GML, KML), y otras utilidades.

Este conjunto de piezas de código, consistente en aproximadamente 3 mil líneas de código PHP, establece las bases para un marco de desarrollo de aplicaciones con características GIS utilizando CodeIgniter.

3.1.1. Requerimientos.

Respecto de los requerimientos de alto nivel, lo fundamental era contar con la implementación en PHP, en particular dentro del *framework* CodeIgniter, de una implementación de *Simple Features*. Esta implementación es el núcleo de un conjunto de librerías que permiten lo siguiente:

- Establecer una base de trabajo para el desarrollo, a través de CodeIgniter, de aplicaciones Web GIS.
- Extender las Aplicaciones Web de Mapas creadas a partir de la aplicación administradora (segundo ítem de este desarrollo), agregándole funcionalidades.

Junto con estos requerimientos de alto nivel, se definen también algunos requerimientos deseables, relacionados con la documentación del código, el apego a estándares *OpenGIS*, el soporte para formatos de intercambio y de representación de información geográfica.

3.1.2. Usuarios.

Los usuarios de estas librerías son programadores con conocimientos en las tecnologías involucradas:

- Programación *web* con PHP, aplicando patrón MVC con CodeIgniter.
- Conocimiento en bases de datos relacionales como PostgreSQL y extensiones espaciales como PostGIS.
- Nociones básicas de MapServer y PHP MapScript.
- Algunas nociones de GIS.

Ellos son los encargados de crear aplicaciones con características GIS, o de extender Aplicaciones Web de Mapas utilizando el conjunto de librerías implementadas.

3.1.3. Clases y librerías implementadas.

El diagrama de clases de las librerías geográficas es el que puede apreciarse en la Figura 3.1. En este modelo se puede ver claramente la separación de las clases en dos grupos: aquellas correspondientes al modelo de clases de *Simple Feature*, y aquellas que proveen de funcionalidades para el trabajo con objetos GIS.

Respecto del modelo de clases de *Simple Features*, este representa los objetos geométricos básicos a partir de los cuales se representan las entidades del mundo real, particularmente en un mapa. Todas las clases heredan de la clase abstracta `Geometry`, que representa un objeto espacialmente referenciado; casi todas, también, tienen asociada una clase de colección.

Respecto de las librerías de utilidades, estas proveen, tanto a la implementación de las *Simple Features*, como al *framework*, de las funcionalidades básicas para el trabajo con datos geográficos. Estas clases fueron programadas e incorporadas al *framework* bajo el esquema de librerías, permitiendo su carga y uso según la necesidad.

Las clases más relevantes de la implementación de *Simple Features* son aquellas utilizadas para almacenar los tipos de información vectorial básica: `Point`, `LineString` y `Polygon`. Tal

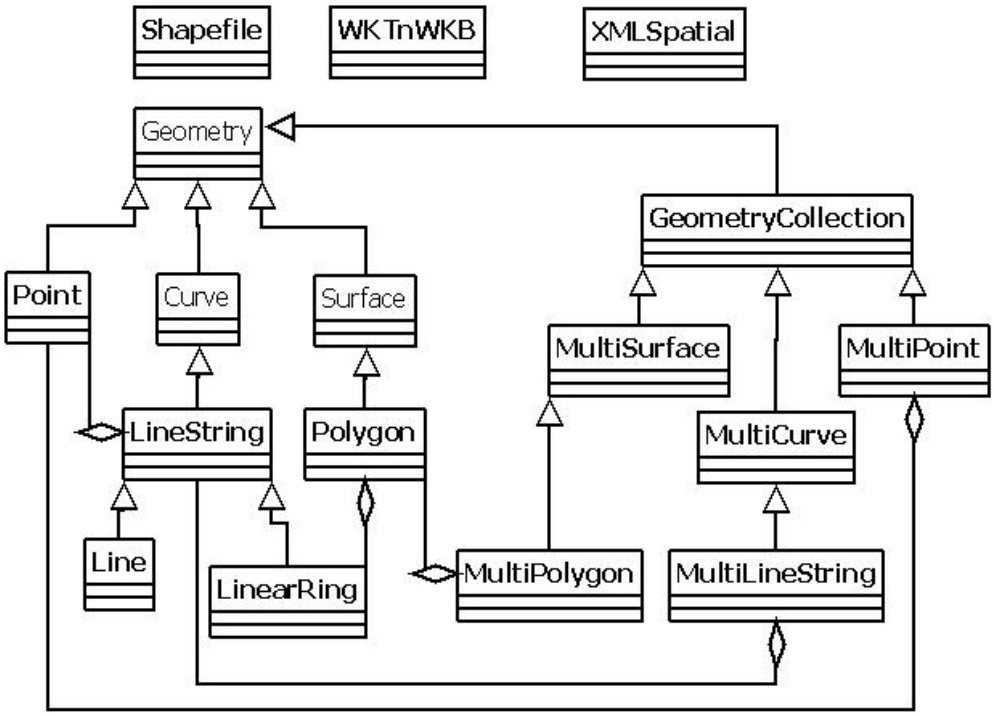

Figura 3.1: Diagrama de clases librerías geográficas.

como se había mencionado antes, todas ellas heredan de la clase abstracta `Geometry` (ya sea directamente o a través de otras abstracciones como `Curve` y `Surface`), y representan objetos espacialmente referenciados (esto es, sus propiedades espaciales, o coordenadas, están asociados a un sistema de referencia particular).

Las clases `Shapefile`, `WKTnWKB` y `XMLSpatial`, implementadas todas bajo el esquema de librerías de CodeIgniter, entregan un conjunto de funcionalidades para el trabajo con los objetos espaciales.

Se describen a continuación las clases más relevantes de la implementación de *Simple Features* para CodeIgniter, y las librerías espaciales implementadas, junto con sus atributos y métodos.

3.1.3.1. Clase `Geometry`.

La clase abstracta `Geometry` encapsula objetos con propiedades espaciales. Sus atributos son:

- `srid`: código del sistema de referencia bajo el cual es representado el objeto espacial. Este sistema de referencia es individualizado por el *geoide*, elipsoide de referencia y proyección cartográfica utilizada; los códigos de cada sistema de referencia son definidos junto con el estándar para *Simple Features* [15].
- `with_m`: *flag* que indica si el objeto posee un atributo *measure*. Este atributo es utilizado para agregar una dimensión al dato espacial (como tiempo o temperatura, por ejemplo), y su valor en cualquier punto de una geometría es interpolado a partir de los valores de sus vértices.

Los métodos más relevantes de esta clase son:

- `is_simple()`: indica si una geometría es simple. Cada subclase de `Geometry` implementa este método, dándole significado según el tipo de geometrá (punto, línea o polígono).
- `is_empty()`: indica si la geometría es el conjunto vacío de puntos.
- `envelope()`: entrega la caja contenedora de la geometría, orientada según los ejes. La caja contenedora es el rectángulo de mínimo tamaño que contiene a la geometría.
- `from_wkt($wkt)`, `from_wkb($wkb)`: construye una geometría a partir de sus representaciones en WKT y WKB, respectivamente.
- `as_wkt()`, `as_wkt()`: entrega las representaciones en WKT y WKB de la geometría, respectivamente.
- `from_georss($georss)`: construye una geometría a partir de su representación en GeoRSS.
- `as_georss()`: entrega la representación GeoRSS de la geometría.
- `from_kml($kml)`: construye una geometría a partir de su representación en KML.
- `as_kml()`: entrega la representación KML de la geometría.
- `from_gml($gml)`: construye una geometría a partir de su representación en GML.
- `as_gml()`: entrega la representación GML de la geometría.

Estos métodos y atributos son hereados (implementados cuando corresponde) al resto de las clases, y conforman el marco base para el trabajo con los objetos espaciales.

3.1.3.2. Clase `Point`.

La clase `Point` representa un punto en $\mathbb{R}^2$. Los objetos de esta clase son utilizados como la primitiva básica con la cual se contruye el resto de los objetos (líneas y polígonos). Los atributos de un punto son:

- `$x`: coordenada X del punto, también llamada Este en el sistema de coordenadas UTM.
- `$y`: coordenada Y del punto, también llamada Norte en el sistema de coordenadas UTM.

Las coordenadas de un punto tienen sentido bajo un sistema de referencia, descrito por el atributo `srid` heredado de `Geometry`.

Los métodos más relevantes de esta clase son:

- `from_coordinates($coords, $srid, $with_m)`: construye un punto a partir de un arreglo con las coordenadas y un sistema de referencia.
- `elipsoidal_distance($point, $a, $b)`: método de instancia que calcula la distancia elipsoidal con otro punto. La distancia elipsoidal entre dos puntos es el arco más corto que los une sobre un elipsoide, descrito por sus semiejes `$a` y `$b` (los valores por omisión corresponden al elipsoide de referencia para WGS 84). Para el cálculo de esta distancia se utiliza el algoritmo de Vincenty, que entrega un resultado con precisión milimétrica (ver Anexo B con la descripción del algoritmo).
- `euclidian_distance($point)`: método de instancia que calcula la distancia euclidiana a otro punto. Para dos puntos $P_1 = [x_1, y_1]$ y $P_2 = [x_2, y_2]$, la distancia euclidiana es la norma de la diferencia: $d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- `get_lon()`: entrega el valor de la latitud del punto sobre el elipsoide de referencia.
- `get_lat()`: entrega el valor de la longitud del punto sobre el elipsoide de referencia.

Los métodos de la clase `Geometry` ralacionados con la representación de puntos en (y a partir de) WKT, WKB, KML, GML y GeoRSS son implementados también.

3.1.3.3. Clase `LineString`.

Esta clase representa una línea formada por una secuencia de puntos. `LineString` hereda de la clase abstracta `Curve` y es extendida por las clases `Line` (que representa una línea recta) y `LinearRing` (que representa una línea simple y cerrada).

Los atributos de la clase `LineString` son:

- `$points`: arreglo de puntos, que corresponden a los vértices de la línea. El arreglo representa en conjunto ordenado de los puntos, construyéndose la línea al unir estos puntos según el orden.

Al estar conformada por puntos, la representación de una línea también toma sentido en función del sistema de referencia en el que se encuentren los puntos.

Los métodos más relevantes de la clase `LineString` son los siguientes:

- `from_points($points, $srid, $with_m)`: construye una línea a partir de un arreglo ordenado de puntos.
- `elipsoidal_length($a, $b)`: método de instancia que calcula el largo de la línea, medida sobre un elipsoide con semiejes `$a` y `$b`. El largo elipsoidal de una línea es entendido como la suma de las distancias elipsoidales entre los puntos consecutivos que conforman la línea.
- `euclidian_length()`: método de instancia que calcula el largo de la línea en el plano cartesiano. Este largo es entendido como la suma de las distancias euclidianas entre los puntos consecutivos que conforman la línea.
- `envelope()`: entrega la caja contenedora de la línea. Las coordenadas de esta caja, en cada uno de los ejes, correponden a los valores máximo y mínimo de las coordenadas de los puntos que conforman la línea.

- `is_closed()`: indica si la línea es cerrada.
- `is_simple()`: indica si la línea no se autointersecta; corresponde a la única restriccón que define una línea válida.

Los métodos de la clase `Geometry` ralacionados con la representación de líneas en (y a partir de) WKT, WKB, KML, GML y GeoRSS son implementados también.

3.1.3.4. Clase `Polygon`.

La clase `Polygon` representa superficies en $\mathbb{R}^2$. Un polígono es representado por un conjunto de anillos, correspondientes a un borde exterior y cero o más bordes interiores. La definición de polígono válido (2.1.3.3) cobra importancia aquí para la implementación del método `is_simple()`.

Los atributos de esta clase son los siguientes:

- `$rings`: arreglo de anillos (líneas simples y cerradas), que corresponden a los bordes del polígono. Se interpreta el primer anillo del arreglo como el borde exterior del polígono, y los siguientes como los bordes interiores.

El arreglo de anillos está compuesto por líneas, cada una compuesta a su vez por puntos, cada uno de los cuales tiene atributos espaciales que tienen sentido en un sistema de referencia dado por el atributo `$srid` heredado de `Geometry`.

Los métodos más relevantes de la clase polígono son los siguientes:

- `from_rings($rings, $srid, $with_m)`: construye un polígono a partir de un arreglo ordenado de anillos que representan sus bordes.
- `area()`: método de instancia que calcula el área del polígono en el plano cartesiano. Para obtener el área se calcula primero el área contenida por cada uno de los anillos (dividiéndo sus interiores en triángulos), y luego al área definida por el borde exterior se le restan las áreas de los bordes interiores.

- `envelope()`: entrega la caja contenedora del polígono. Las coordenadas de esta caja, en cada uno de los ejes, corresponden a los valores máximo y mínimo de las coordenadas de los puntos que conforman el borde exterior del polígono.

- `centroid()`: entrega el punto correspondiente al centroide matemático del polígono. Dado que los polígonos no convexos pueden ser polígonos válidos, el centroide puede caer fuera de la figura.

- `is_simple()`: indica si el polígono es válido, según las restricciones establecidas en 2.1.3.3.

Los métodos de la clase `Geometry` ralacionados con la representación de polígonos en (y a partir de) WKT, WKB, KML, GML y GeoRSS son implementados también.

3.1.3.5. Librería `Shapefile`.

Esta librería tiene como propósito implementar las funcionalidades necesarias para trabajar con archivos *shapefile*. La implementación se justifica puesto que los *shapefiles* son los archivos más utilizados para el almacenamiento de información geográfica en formato vectorial.

Los principales métodos de esta librería son los siguientes:

- `get_shp_properties($shapefile)`: entrega un arreglo (*hash*) con las propiedades globales de un *shapefile*, tales como sistema de referencia, elipsoide o datum, tipo de geometrías (punto, línea o polígono) y número de registros.

- `get_shp_extent($shapefile)`: obtiene la extensión del *shapefile*, entendida como la caja contenedora de todas las geometrías en el archivo.

- `read_shp($shapefile)`: carga en un arreglo de geometrías (`GeometryCollection`) los objetos geográficos y en una tabla la lista de atributos de cada objeto.

- `write_shp($geoms, $atts, $shapefile)`: dado un conjunto de geometrías y una tabla de atributos para estas, crea un archivo *shapefile*.

3.1.3.6. Librería `WKTnWKB`.

Esta librería entrega las funcionalidades necesarias para trabajar con las representaciones en WKT y WKB de las geometrías. Los métodos más relevantes de esta clase son los siguientes:

- `parse_wkt($wkt)`: este método toma un string en formato WKT y retorna la geometría que representa.
- `get_wkt($geom)`: este método toma una geometría y retorna un string con su representación en WKT.
- `parse_wkb($wkb)`: este método toma un arreglo de bytes en formato WKB y retorna la geometría que representa.
- `get_wkb($geom)`: este método toma una geometría y retorna un arreglo de bytes con su representación en WKB.
- `wkt_to_wkb($wkt)`, `wkb_to_wkt($wkb)`: toma la representación de una geometría en WKT (WKB) y entrega su representación en WKB (WKT).

3.1.3.7. Librería `XMLSpatial`.

Esta librería implementa los métodos necesarios para el trabajo con los diferentes formatos de intercambio de información considerados: GML, KML y GeoRSS. Para cada tipo de formato se implementa un *parser* que lee el documento XML y permite extraer los datos espaciales para cargarlos en colecciones de geometrías; por cada tipo de formato, también, se implementa un método para generar un documento XML a partir de un conjunto de geometrías.

Los principales métodos de esta librería son:

- `load_gml($gml)`, `load_kml($kml)`, `load_georss($georss)`: métodos que leen (utilizando *parsers*) un archivo GML, KML o GeoRSS, y cargan una colección de geometrías con los datos que éstos traen.
- `write_gml($geoms)`, `write_kml($geoms)`, `write_georss($geoms)`: dado un conjunto de geometrías, estos métodos generan un archivo GML, KML o GeoRSS.

3.2. Aplicación administradora.

El objetivo principal de esta aplicación es generar, de manera simple y transparente, Aplicaciones Web de Mapas en pocos pasos. Esto permite que los administradores de la información geográfica la hagan pública al resto de la organización, de manera ordenada y controlada.

El desarrollo de esta aplicación se llevó a cabo siguiendo una metodología en cascada, y constó de cuatro etapas:

- Análisis de Requerimientos: en esta etapa se definieron los requerimientos de la aplicación, principalmente basados en la herramienta ArcGIS Manager de ArcGIS Server.
- Diseño: en esta etapa se definió el modelo de datos, las componentes de la aplicación y sus interfaces.
- Implementación: en esta etapa se codificó la aplicación.
- Pruebas: en esta etapa se validaron los compromisos adquiridos.

3.2.1. Análisis de Requerimientos.

3.2.1.1. Usuarios.

Esta aplicación está enfocada a los administradores de la información geográfica dentro de una organización. Aunque el perfil de estos usuarios varía dependiendo del sector de la industria, en términos generales se trata de profesionales con las siguientes características:

- Profesionales de las Geociencias con conocimientos en Geomática (geógrafos, cartógrafos).
- Dominio de herramientas GIS de escritorio (Quantum GIS) y *web*.
- Conocimientos en administración de una *geodatabase*.

Estos usuarios son los responsables de mantener la información geográfica ordenada y estandarizada, y de controlar los procesos de entrada y salida de información del GIS. Para

ellos, la herramienta desarrollada automatiza de un modo simple la publicación ordenada, y sujeta a los privilegios que definan, de la información que administran.

3.2.1.2. Requerimientos de alto nivel.

Casi todos los requerimientos para esta aplicación, se originan en la necesidad de contar con una herramienta similar a ArcGIS Manager de ESRI, pero de bajo costo y utilizando herramientas de código abierto. Hay que recalcar que esta necesidad es real, y ha sido expresada en diversas licitaciones públicas por parte de municipios y otros organismos de gobierno[1]. Tales organizaciones, si bien requieren administrar información geográfica, no cuentan con recursos para sostener plataformas tan costosas como la de ESRI.

Los requerimientos de alto nivel de la aplicación administradora de mapas son los siguientes[2]:

- La aplicación debe instalarse a través de una interfaz *web* que permita realizar la configuración de los parámetros iniciales en no más de 5 pasos simples. Una vez terminado este proceso, la herramienta debe quedar lista para ser utilizada.
- Una vez correctamente instalada y configurada la aplicación, los usuarios deben poder publicar la información geográfica de la que disponen, previa organización de esta en un archivo *mapfile*. La información geográfica puede provenir de tres distintos tipos de orígenes: archivos (*shapefiles* principalmente), base de datos y WMS. La forma en que se publiquen estos datos será la de una Aplicación Web de Mapas.
- En régimen normal, los usuarios podrán administrar (editar y eliminar, cambiar el estilo) las Aplicaciones Web de Mapas creadas a partir de la herramienta administradora.
- Las Aplicaciones Web de Mapas creadas a través de esta aplicación deben poder modificarse individualmente de acuerdo a necesidades específicas (extensiones), y utilizando las librerías destinadas para ello.

[1] Licitación 5482-154-LP09 en Mercado Público: Implementación de un Servicio de Información Geográfica, para la Municipalidad de Ñuñoa.

[2] Un listado de requerimientos más detallado, y clasificados por tipo, pueden verse en el Anexo A .

La Figura 3.2 ilustra los casos de uso de alto nivel desprendidos de los requerimientos descritos en el punto anterior; los Cuadros 3.1, 3.2, 3.3 y 3.4 describen brevemente estos casos de uso.

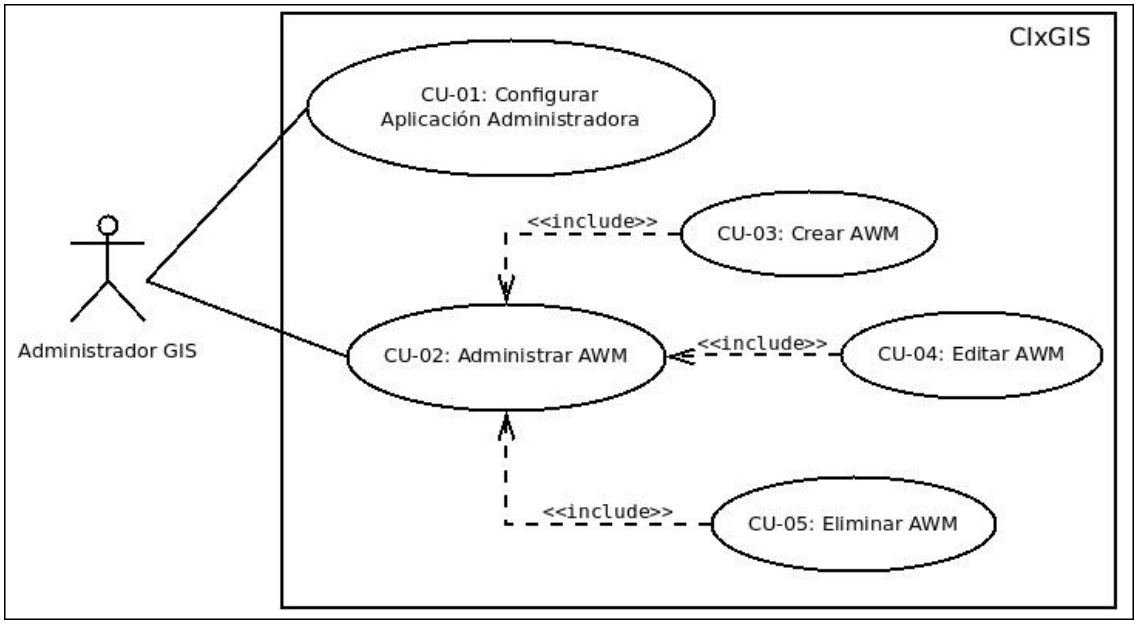

Figura 3.2: Diagrama de Casos de Uso de alto nivel.

Nombre	CU-01: Configurar Aplicación Administradora
Propósito	Realizar las configuraciones iniciales a la aplicación, que la habiliten para ser operada.
Actores	Administrador
Descripción	Una vez desplegada la aplicación, el usuario ejecuta una página de instalación que le permite ingresar los parámetros básicos de configuración (coexiones a bases de datos, rutas de archivos, entre otros) de la herramienta.

Cuadro 3.1: Caso de uso CU-01.

3.2.2. Diseño.

3.2.2.1. Modelo de Datos.

El modelo de datos que soporta los requerimientos de la aplicación es el que se puede apreciar en el diagrama Entidad-Relación (Figura 3.3) y en el diagrama Físico de Datos (Figura 3.4).

Las entidades más relevantes de este modelo son:

Nombre	CU-03: Crear AWM
Propósito	Crear una Aplicación Web de Mapas.
Actores	Administrador
Descripción	El usuario ingresa a la administración de AWM, donde puede crear una nueva. Ingresando parámetros como el origen de datos, nombre del sitio, entre otras, crea una nueva AWM.

Cuadro 3.2: Caso de uso CU-03.

Nombre	CU-04: Editar AWM
Propósito	Editar propiedades de una Aplicación Web de Mapas.
Actores	Administrador
Descripción	El usuario ingresa a la administración de AWM, donde puede listar las AWMs sexistentes. Seleccionando una, puede acceder a sus propiedades y modificarlas.

Cuadro 3.3: Caso de uso CU-04.

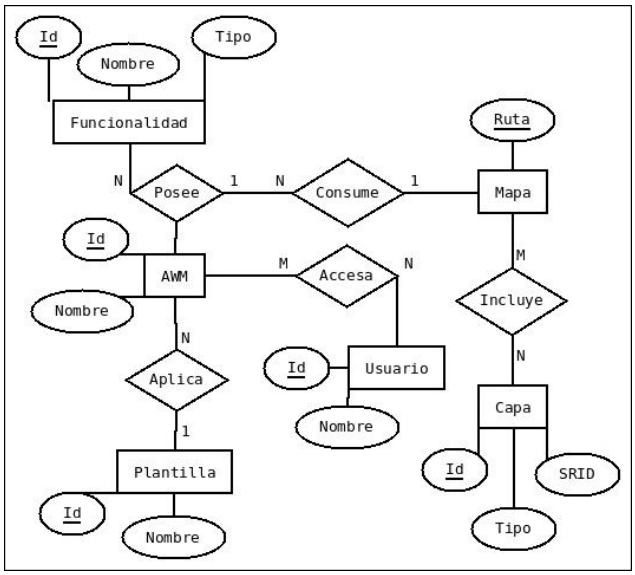

Figura 3.3: Diagrama Entidad-Relación.

Nombre	CU-05: Eliminar AWM
Propósito	Eliminar una Aplicación Web de Mapas.
Actores	Administrador
Descripción	El usuario ingresa a la administración de AWM, donde puede listar las AWMs sexistentes. Seleccionando una, puede eliminarla.

Cuadro 3.4: Caso de uso CU-05.

- Aplicación Web de Mapas (AWM): representa una Aplicación Web de Mapas que publica información disponible y estructurada a patrir de un documento *mapfile* (entidad Mapa).

- Mapa: representa un documento *mapfile* que toma información geográfica de diferentes fuentes (archivos, base de datos, WMS), la estructura en capas y les asigna propiedades básicas (como proyección, colores, extensión, etc.).

- Capa: representa una capa de información geográfica (de Puntos, Líneas o Polígonos), incluída dentro de un documento de mapas.

- Plantilla: representa un estilo que aplica a las Aplicaciones Web de Mapas, compuesto por hojas de estilo (CSS), imágenes, etc.

- Usuario: representa a los usuarios que tienen acceso a las Aplicaciones Web de Mapas generadas a través de la aplicación.

- Funcionalidad: representa una funcionalidad encapsulada en un grupo de funciones programadas y que pueden ser agregadas a las Aplicaciones Web de Mapas una vez creadas. Ejemplos de funcionalidades son: acercamientos al mapa, despliegue de atributos de los objetos en el mapa, medición de distancias, etc.

Una de las ventajas de CodeIgniter (al igual que la mayoría de los *frameworks* MVC) es que abstrae la capa de acceso al repositorio de datos. Esto permite que las aplicaciones desarrolladas con el *framework* sean fácilmente extensibles para trabajar con distintos motores de datos, entre los cuales se cuentan MySQL, MySQLite, PostgreSQL y SQL Server. Aunque en un principio la aplicación administradora fue diseñada para trabajar con PostgreSQL (extendido con PostGIS para manejar datos espaciales), no es muy complejo realizar los ajustes necesarios para que lo haga con MySQL (extendido con MySQL Spatial).

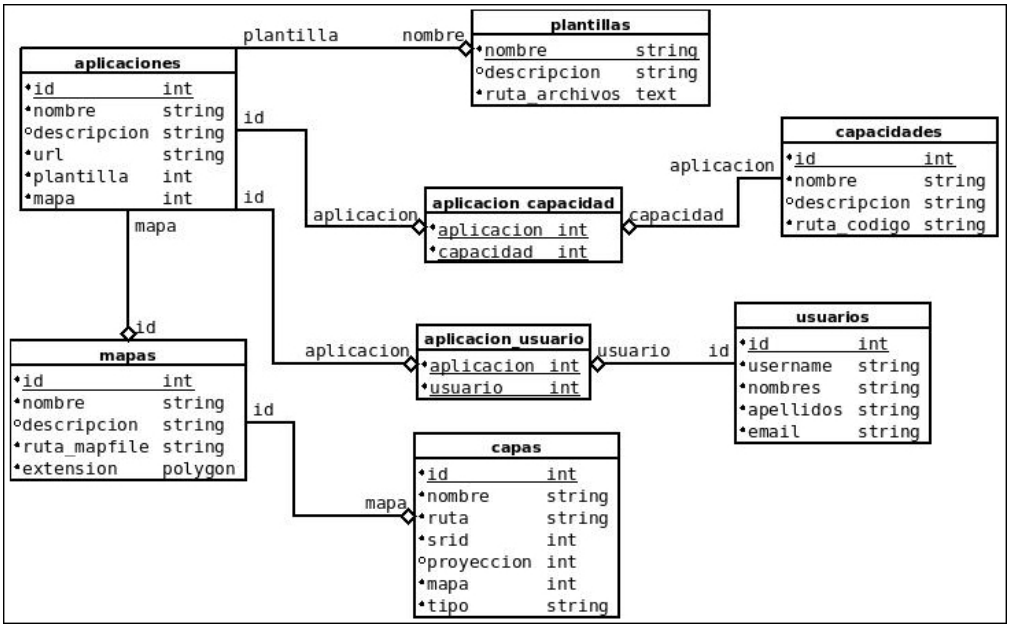

Figura 3.4: Diagrama físico de la base de datos.

3.2.2.2. Modelo de Clases.

Dado que CodeIgniter aplica el patrón MVC, las clases para esta aplicación fueron implementadas bajo los estereotipos de controlador y modelo.

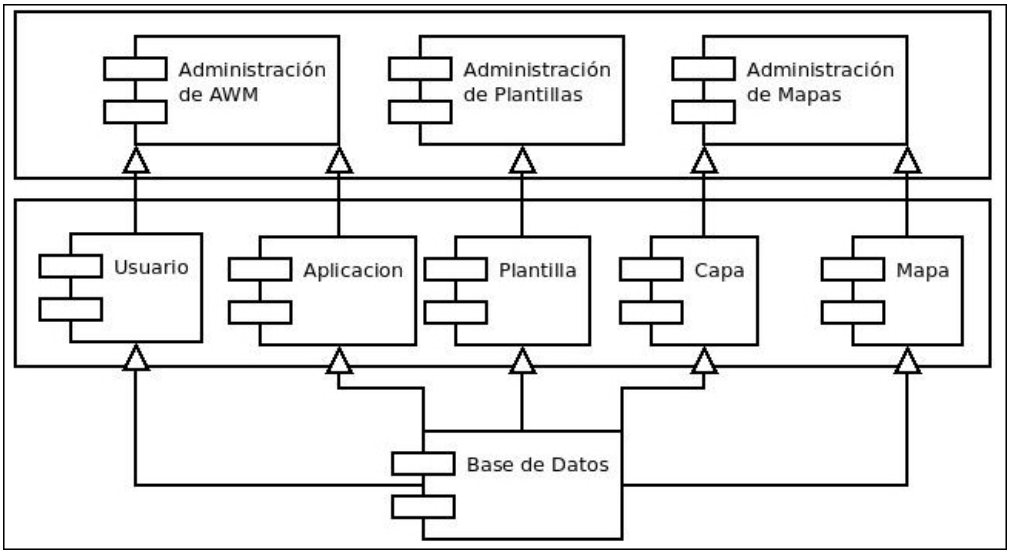

Figura 3.5: Diagrama de componentes de la aplicación.

Las clases del modelo se desprenden directamente del modelo de datos: `Aplicacion`, `Plantilla`, `Mapa`, `Capa` y `Usuario`. Estas clases proveen de los métodos CRUD para almacenamiento y consulta en la base de datos.

Las clases controladoras, por otro lado, son las encargadas de manejar el flujo de la aplicación para proveer las funcionalidades a los usuarios. Podemos considerar una agrupación de ellas en tres módulos, como puede observarse en la Figura 3.5. Tal como lo establece el patrón MVC, las clases controladoras utilizan las clases (módulos) del modelo, además de las vistas, para atender las peticiones de los usuarios.

Las ventajas de la elección de un *framework* MVC pueden apreciarse en el momento de diseñar e implementar la aplicación administradora: la modularidad, la separación de funciones, la claridad y simpleza de cada uno de los módulos y clases, hacen de ésta una aplicación fácilmente extensible y mantenible en el tiempo.

3.2.3. Implementación.

Para la implementación se tomó como base el *framework* extendido con las librerías geográficas. Esto demuestra que las extensiones por sí solas constituyen un producto que permite el desarrollo de aplicaciones Web GIS.

Conforme al patrón MVC que implementa CodeIgniter, se crearon tres tipos de componentes: vistas, controladores y clases del modelo.

3.2.3.1. Vistas.

Las 18 vistas que forman parte de la aplicación de administración fueron programadas separando la estructura en HTML de los estilos en CSS. Se utilizó además un conjunto de librerías en javascript que ejecutan, del lado del cliente, un conjunto de funcionalidades que enriquecen las interfaces *web* (Scriptaculous). Estas vistas conforman la presentación de los datos de la aplicación de administración, para cada uno de sus módulos: Administración de AWM, Mapas, Capas, Usuarios y Priviegios y Plantillas.

Durante la programación de las vistas se privilegió la conformidad con los estándares *web* del W3C para HTML y CSS. En este sentido, los validadores de código HTML y CSS[3] guiaron la programación de las vistas de la aplicación.

[3]Herramientas de validación *web* disponibles en `http://validator.w3.org/` y `http://jigsaw.w3.org/css-validator/`.

Se minimizó, en la medidad de lo posible, el código PHP embebido en las vistas, de modo que no se incluyera lógica de aplicación donde no corresponde.

Todas las vistas fueron construidas a partir de una pantilla de 4 bloques de contenido: encabezado, menú, contenido principal y pie de página; esta organización, aparte de modularizar los contenidos, permite una gran flexibilidad al momento de cambiar el estilo de presentación.

3.2.3.2. Controladores.

Se implementaron 6 controladores que prestan los servicios básicos descritos en la Figura 3.5. Estos controladores contienen las funciones básicas para la interacción del usuario (recogida desde las vistas) con los datos, además del flujo de las operaciones básicas de la aplicación. Para cada controlador se programaron los siguientes grupos de funciones comunes:

- Listar: estas funciones acceden al modelo de datos correspondiente para listar, según algún criterio de filtro establecido por el usuario, los objetos en la base de datos (AWM, plantillas, mapas, etc.).
- Crear: estas funciones permiten a los usuarios la creación de objetos del modelo, previa verificación de los datos (consistencia, valores válidos, etc.).
- Editar: estas funciones permiten acceder a los elementos del modelo y editar sus propiedades.
- Eliminar: estas funciones permiten la eliminación de elementos del modelo, incluyendo validaciones previas para cada caso, con el fin de mantener la integridad de los datos.

Existe un principio de correspondencia entre las funciones de cada controlador y las vistas. Esta regla, que por cada función en un controlador define una vista para la interacción con el usuario, permite que sea simple agregar nuevas funcionalidades a los controladores sin tener que modificar las vistas existentes.

Los controladores fueron programados en PHP, y heredan de la clase base `Controller` de CodeIgniter. Son estas clases las que acceden mayormente a las librerías del *framework*, en particular a las librerías geográficas implementadas como primera parte del desarrolo.

3.2.3.3. Modelo.

Las clases del modelo corresponden a 6 archivos de aproximadamente 150 líneas de código PHP cada uno, que proporcionan los métodos CRUD para el acceso y consulta a la base de datos. Según el patrón MVC que implementa CodeIgniter, estas clases heredan de la clase `Model` del *framework*, y representan a los objetos (persistentes en un repositorio de datos en este caso) que son manejados por la aplicación administradora.

A excepción de aquellas que almacenan información del sistema, cada tabla en la base de datos tiene una clase del modelo asociada. Todas las clases implementan al menos las siguientes funciones para el acceso a los datos:

- `get_all()`: obtiene la lista completa de objetos (un *recordset*) en la tabla del repositorio de datos.
- `get_list()`: obtiene la lista completa de objetos (un hash indexado con el id del objeto) en la tabla del repositorio de datos.
- `get($id)`: obtiene un objeto particular a partir de su código en la base de datos.
- `delete($id), save($id), update($id)`: métodos para eliminar, guardar y actualizar un objeto en la base de datos.

Cuando corresponde, hay una validación de los datos antes de insertar, actualizar o eliminar el registro en una tabla.

3.2.4. Pruebas.

Las pruebas a la aplicación administradora tuvieron como propósito principal determinar la consistencia de la aplicación desarrollada con los requerimientos planteados, y entregar pautas para ajustes y correcciones a la aplicación. A grandes rasgos, se consideraron los dos grupos de prueba resumidos en el Cuadro 3.5.

Todas estas pruebas fueron cumplidas por la aplciación desarrollada. La complejidad cilomática

Tipo	**Objetivo**	**Pruebas**
Funcionalidad (caja negra)	Verificar el cumplimiento de los requerimientos funcionales de la aplicación.	- Pruebas de las funciones de controladores y clases del modelo ante condiciones de borde en las entradas. - Creación, edición y eliminación de AWM. - Creación, edición y eliminación de Mapas y Capas. - Creación, edición y eliminación de Plantillas. - Aplicar una plantilla a una AWM. - Creación, edición y eliminación de usuarios. - Asignar y quitar permisos en AWM a los usuarios.
Diseño (caja blanca)	Verificar la correcta programación y diseño de los módulos de la aplicación.	- Pruebas de complejidad ciclomática de los controladores. - Pruebas de complejidad ciclomática de las clases del modelo.

Cuadro 3.5: Pruebas aplicadas.

en cada uno de los módulos de la aplicación fue siempre menor a 5, lo que valida la simpleza de éstos.

Además se aplicaron mediciones de accesibilidad y usabilidad de las interfaces *web* de la aplicación, validando las siguientes propiedades, transversalmente dentro de la aplicación:

- Menú de operaciones completo disponible dentro de todas las interfaces de la aplicación.
- No más de 2 clicks para acceder a una funcionalidad de la aplicación.
- Descripción, a través de páginas de ayuda, de cada una de las operaciones provistas por la aplicación.
- Descripción de cada valor que debe ser ingresado por los usuarios (a través de *tooltips*).
- Ninguna interfaz de la aplicación presenta *scroll* para una resolución de 1024x768 pixeles.

Capítulo 4

Validación.

Este capítulo corresponde a la validación de los resultados obtenidos en el desarrollo de la aplicación administradora y generadora de Aplicaciones Web de Mapas. Se relata una historia de uso para la publicación de datos geográficos disponibles en una organización ficticia de gobierno regional.

Se describe el contexto dentro del cual se enmarca la problemática, para luego proponer una solución. Finalmente se describen los pasos necesarios para implementar dicha solución:

- Instalación de la aplicación administradora.
- Preparación de la información.
- Creación de una Aplicación Web de Mapas.

Se verifica, a través de este proceso de publicación de información geográfica, el cumplimiento de las funcionalidades requeridas para el producto final de este trabajo.

4.1. Contexto.

Una organización de gobierno regional, que llamaremos Minsal, es la encargada de fiscalizar a los establecimientos que prestan diversos servicios que tienen algún impacto en la salud de la población y el medioambiente. Ejemplos de establecimientos tales son:

- Establecimientos que mantengan productos químicos en bodega, como fábricas de pintura.
- Establecimientos que procesen alimentos para la población, como carnicerías.
- Establecimientos que presten servicios sanitarios básicos, como peluquerías y solariums.

La fiscalización de estos establecimientos tiene como objetivos tanto la prevención de riesgos sanitarios, como la elaboración de planes de respuesta ante emergencias.

Para estos fines, el Minsal ha catastrado los establecimientos afectos a fiscalización, incluyendo los datos geográficos de estos. La componente geográfica cumple un papel fundamental por cuanto ayuda a la correcta planificación de las visitas de los inspectores y apoya el análisis territorial (zonas de concentración de bodegas con químicos, por ejemplo).

Sin embargo, aún cuando la información geográfica existe, no puede ser accedida por las distintas áreas del Minsal. Sólo los administradores de datos disponen de la información, transformándose en un cuello de botella para el acceso del resto de la organización.

4.2. Solución propuesta.

La solución propuesta es la implementación de una base de datos geográfica (una *geodatabase*), que centralice la información que dispone el Minsal de los establecimientos afectos a fiscalización sanitaria, además de otro tipo de información geográfica que pueda disponer a futuro. Junto con la implementación de la *geodatabase*, se propone también la instalación de la herramienta de generación de Aplicaciones Web de Mapas. Esta permitirá a los administradores de los datos, divulgar estos de manera ordenada al resto de su organismo.

Para la implementación de esta solución, dadas las dimensiones de la organización, se necesitará en un comienzo de una sola máquina que ejecute los servicios de base de datos y de aplicaciones *web*. Un servidor de capacidad media[1] será suficiente; la solución a implementar permite escalar el *hardware* en el futuro, si es necesario.

Respecto del *software*, se propone una configuración básica para soportar la instalación y operación de la herramienta de generación de mapas *web*:

- Sistema Operativo Linux (Debian 5.0.5 Lenny).
- Servidor *web* Apache 2.0.
- PHP 5 y librería PHP MapScript.
- Servidor de base de datos PostgreSQL 8.4, con las extensiones espaciales PostGIS.
- Servidor de Mapas MapServer 5.6.1.

Tanto el sistema operativo como las versiones de los distintos paquetes de *software* mencionados, si bien no son exclusivos[2], son los recomendados pues constituyeron el ambiente de desarrollo y pruebas de la aplicación.

4.3. Instalación y configuración inicial.

Para la instalación, basta copiar los archivos fuente al directorio de sitios de Apache y acceder a la raíz del sitio. Un `script` de instalación se ejecutará en tres pasos (ver Figuras 4.1, 4.2 y 4.3, con las capturas de pantalla de las interfaces para estos pasos):

- Configuración de las base de datos: opciones para escoger el tipo de base de datos, y los datos de conexión (usuario, contraseña, etc.).

[1] Procesador de 2 GHz, 100 Gb de disco duro y 4 Gb de RAM.

[2] La aplicación puede desplegarse en una máquina con sistema operativo Windows, si ésta ejecuta Apache, PHP, PostgreSQL y MapServer; una distribución de Quantum GIS para Windows incluye todos estos componentes para ser instalados opcionalmente.

- Configuración de rutas: rutas para los archivos de mapas (*mapfiles*), plantillas, archivos de datos, imágenes raster, etc.
- Configuración inicial de privilegios: datos para el usuario administrador.

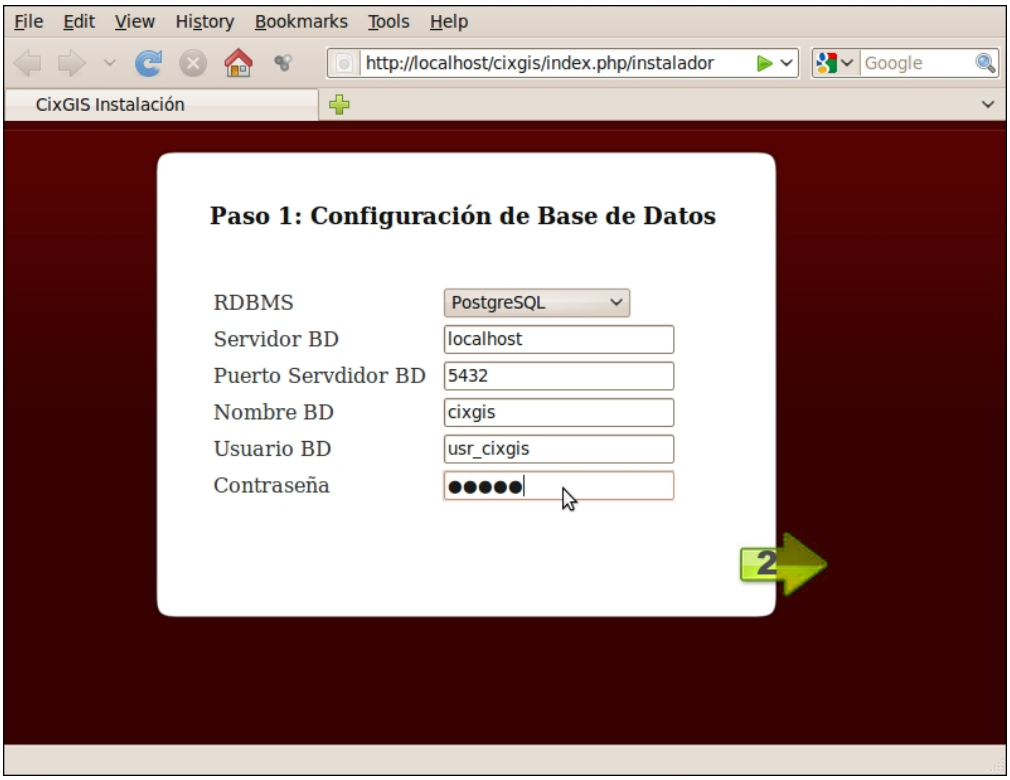

Figura 4.1: Interfaz de instalación, Paso 1.

Con estos simples pasos, ya se dispone de una herramienta para publicar en *web* la información geográfica disponible.

4.4. Preparación de la información.

El Minsal dispone de una planilla Excel con el catastro de los establecimientos, incluyendo sus coordenadas. Sin embargo, los puntos por sí solos no tienen mucha utilidad, por lo que hay que ubicarlos dentro de un contexto geográfico.

Se consideran, entonces, las siguientes capas de información geográfica, que deberán ser preparadas para la carga en la *geodatabase*:

- Establecimientos: capa de puntos que representan los establecimientos.

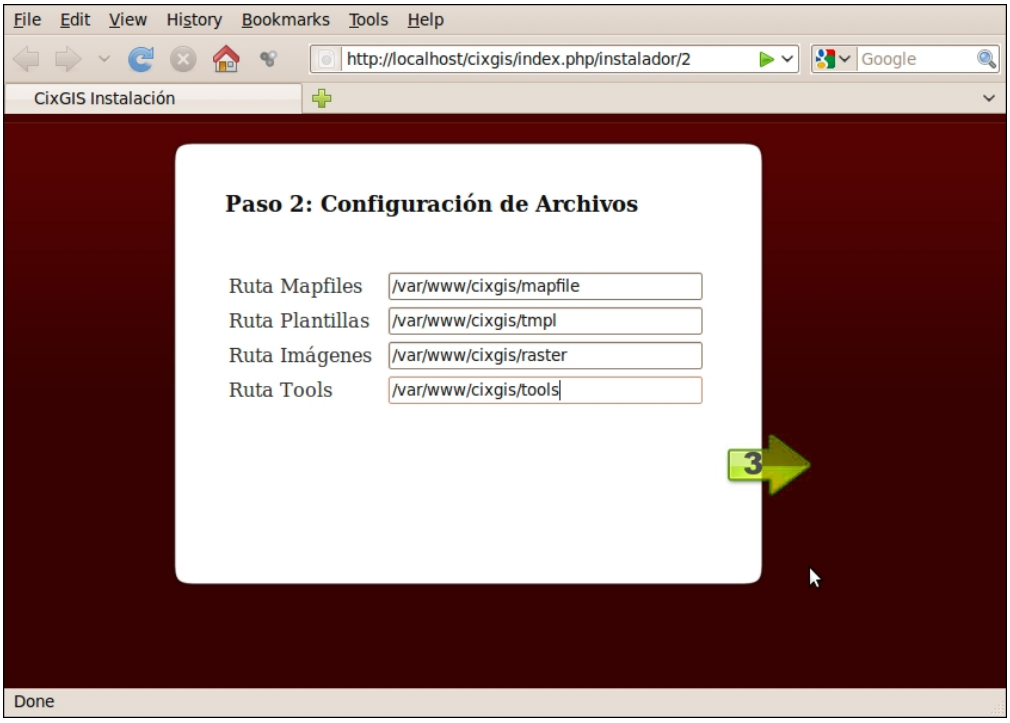

Figura 4.2: Interfaz de instalación, Paso 2.

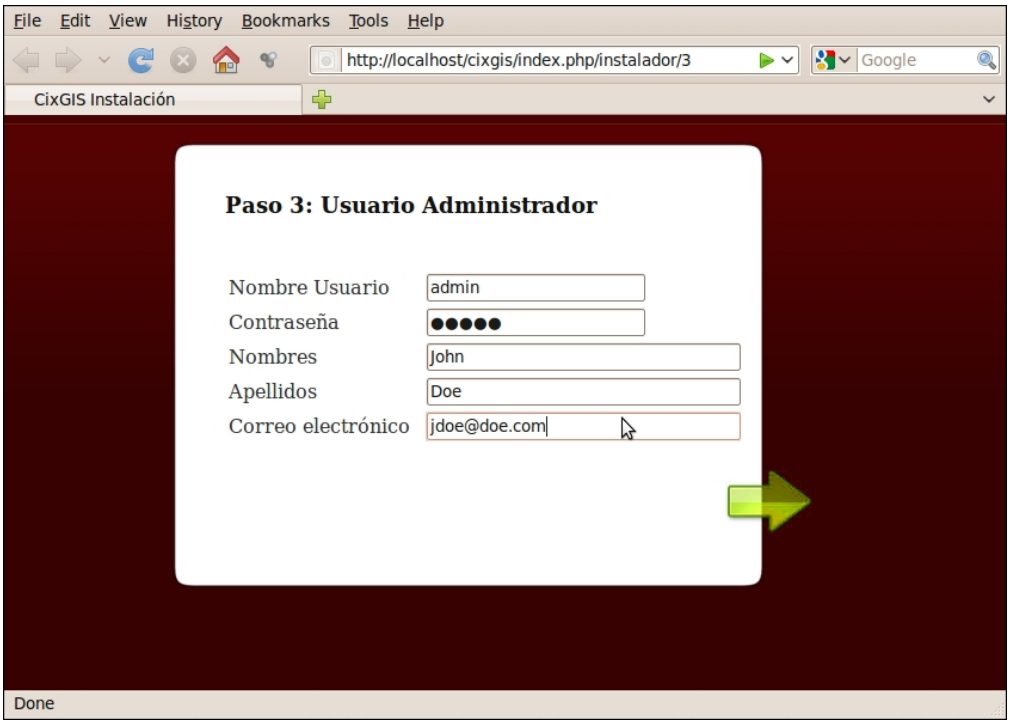

Figura 4.3: Interfaz de instalación, Paso 3.

- Comunas: capa de polígonos que representan los límites comunales dentro de la Región del Maule.
- Caminos: capa de líneas que representan los caminos dentro de la región.
- Ríos: capa de líneas que representan los cursos de agua en la región.

Las capas de Comunas, Caminos y Ríos son obtenidas a través de un organismo estatal. La capa de Establecimientos es construida a partir del archivo Excel que dispone el Minsal. Para su construcción, se aplica un procedimiento de carga de datos utilizando el *software* GIS de escritorio QGIS. Se genera un archivo *shapefile* que es cargado directamente en la *geodatabase* utilizando el mismo QGIS.

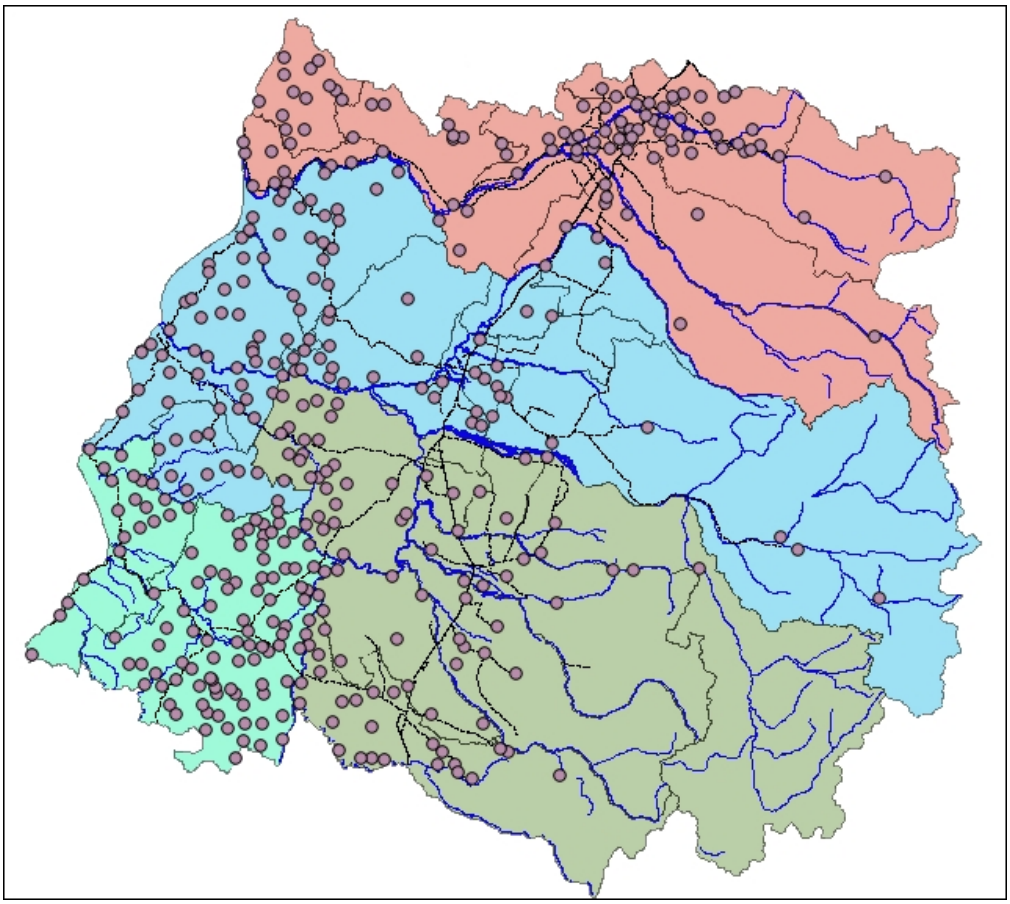

Figura 4.4: Vista de capas geográficas en QGIS.

Toma más sentido, así, la información de los establecimientos (la Figura 4.4 muestra el mapa resultante de la combinación de estas capas con QGIS).

4.5. Creación de una Aplicación Web de Mapas.

Para la creación de la aplicación *web* que publica la información geográfica, es necesario primero disponer de un archivo *mapfile* que describa las capas que conforman el mapa, junto con algunas propiedades como el sistema de proyección, los colores de las capas, el origen de cada una de ellas, y las leyendas o simbologías incluidas. Un archivo *mapfile* que incluye las capas de comunas y establecimientos luce como sigue:

establecimientosmaule.map

```
MAP
    NAME Establecimientos Maule
    SIZE 700 500
    STATUS ON
    EXTENT 158440 5952900 381027 6157400

    FONTSET "misc/fonts/fonts.txt"
    SYMBOLSET "misc/symbols/symbols.sym"

    IMAGECOLOR 255 255 255
    UNITS dd

    WEB
        IMAGEPATH "/var/www/cixgis/tmp/"
        IMAGEURL "tmp/"
    END

    LAYER
        CONNECTIONTYPE postgis
        NAME "Comunas"
        STATUS ON

        CONNECTION "user=postgres dbname=cixgis host=localhost"
        DATA "the_geom FROM COMUNAS as poligono using unique gid using SRID=-1"

        TYPE POLYGON
        CLASS
            STYLE
                COLOR 255 123 0
                OUTLINECOLOR 0 0 0
            END
        END
    END

    LAYER
        CONNECTIONTYPE postgis
        NAME "Establecimientos"
        STATUS ON

        CONNECTION "user=postgres dbname=cixgis host=localhost"
        DATA "the_geom FROM ESTABLECIMIENTOS as poligono using unique gid using SRID=-1"

        TYPE POINT
        CLASS
            STYLE
                COLOR 0 0 0
                OUTLINECOLOR 0 0 0
            END
        END
    END
END
```

Una vez creado este archivo, se carga a través de la aplicación administradora, que lo registra (junto con las capas que posee) y lo deja en la carpeta correspondiente. Luego de realizar esta tarea, estamos en condiciones de crear la Aplicación Web de Mapas. Las Figuras 4.5 y 4.6 muestran las interfaces para la creación de una nueva Aplicación Web de Mapas (ingreso de datos) y para el despliegue de las ya existentes.

Esta aplicación generada (como se puede ver en el ejemplo de la Figura 4.7) será la base para extensiones específicas que se necesiten. En este caso particular, puede resultar interesante,

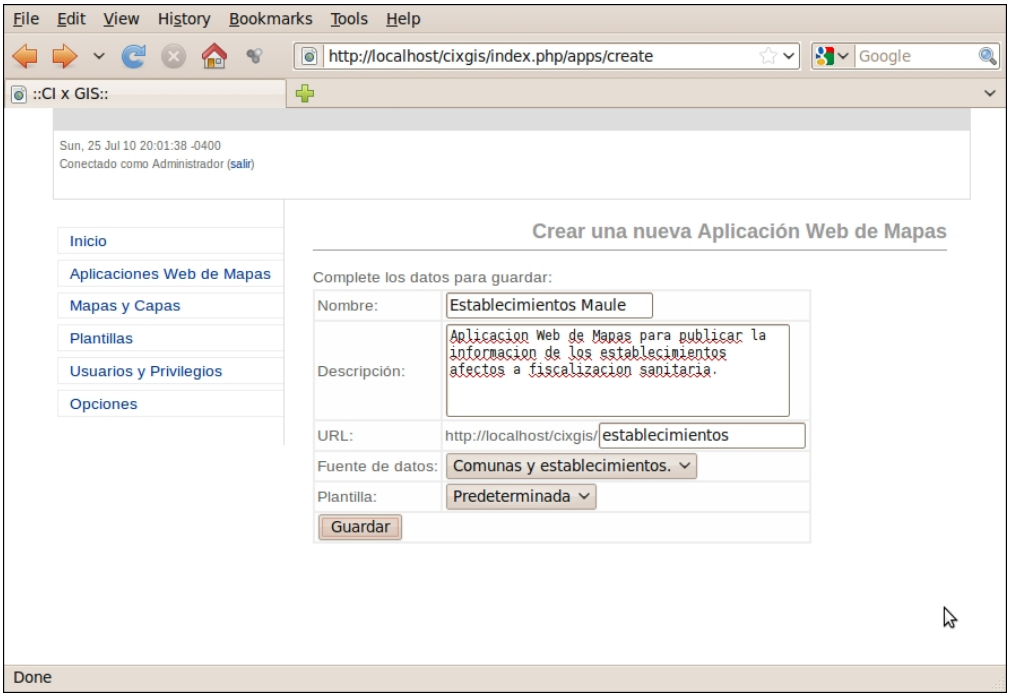

Figura 4.5: Creación de una Aplicación Web de Mapas.

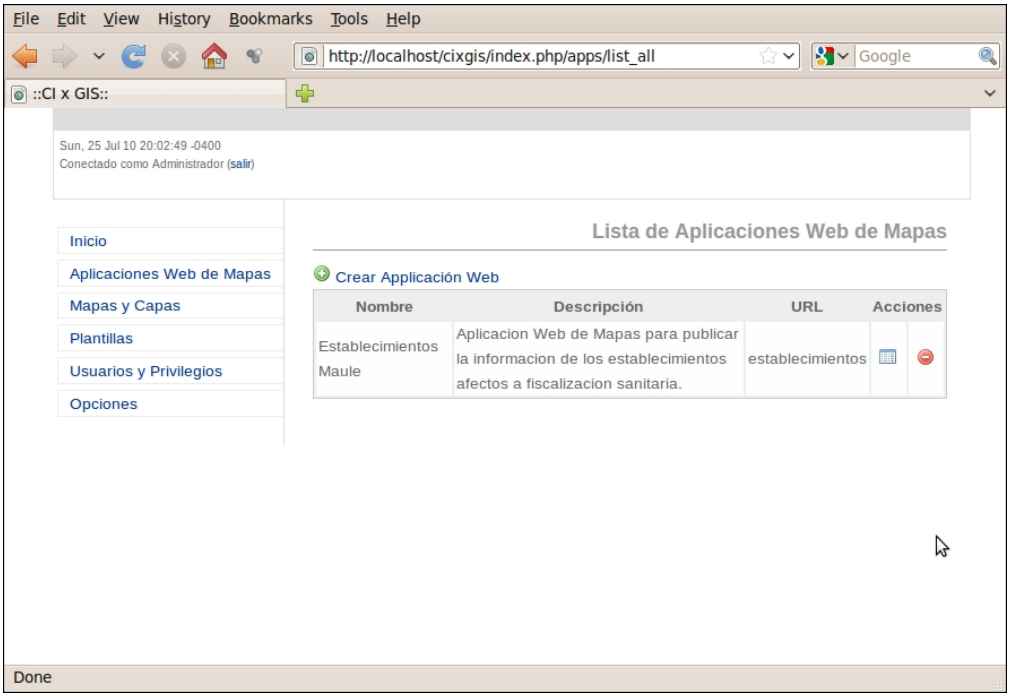

Figura 4.6: Lista de Aplicaciones Web de Mapas.

por ejemplo, mostrar un cuadro con las propiedades de los puntos al seleccionarlos en el mapa.

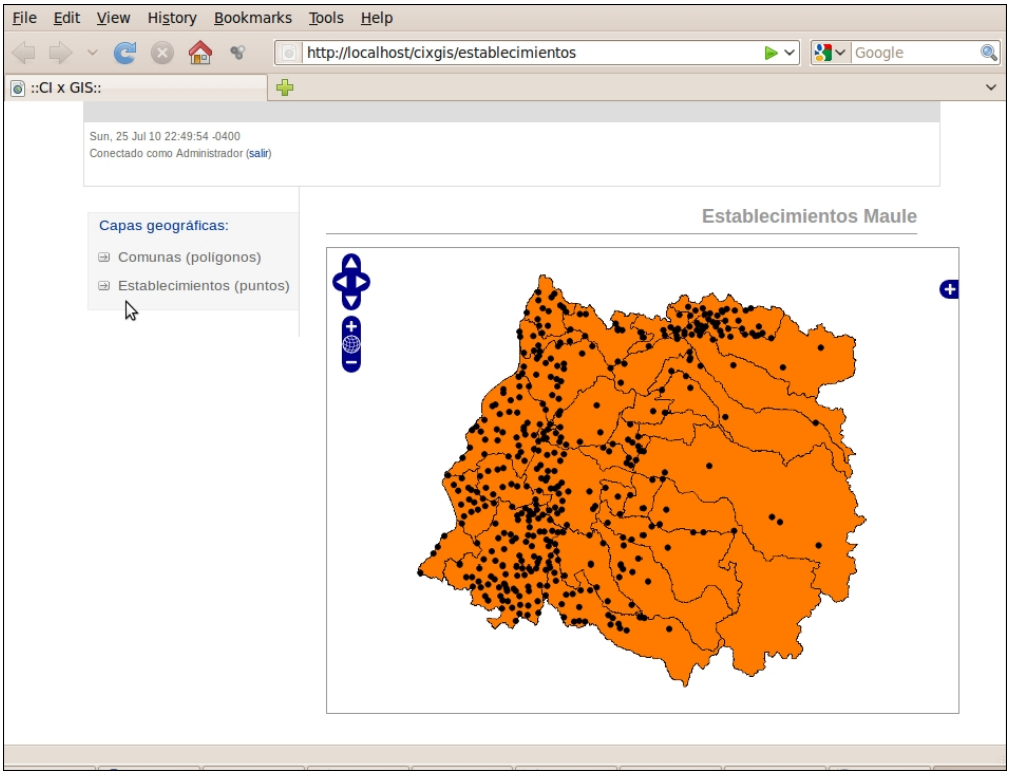

Figura 4.7: Aplicación Web de Mapas generada.

Capítulo 5

Conclusiones.

En este capítulo se presentan las conclusiones al trabajo realizado.

Primero se presentan las conclusiones obtenidas al final de este trabajo, en particular acerca del producto obtenido y dentro del marco de las necesidades que gatillaron el desarrollo.

Finalmente se plantean sugerencias acerca de cómo encaminar los resultados obtenidos, y se mencionan algunas recomendaciones y tareas pendientes.

5.1. Conclusiones sobre resultados obtenidos.

La información geográfica se ha convertido en un activo altamente valioso en muchas organizaciones de distinto tamaño, gatillando la necesidad de un GIS[1]. Dentro de esta implementación, en el final de la cadena del tratamiento de la información, está la publicación de la misma. Es en este último eslabón donde este trabajo entrega su aporte.

Existen las herramientas de *software* para cubrir la necesidad de publicar la información geográfica, de una manera ordenada y controlada; proveedores como ESRI o AutoDesk han avanzado en este sentido. Sin embargo el acceso a estos avances es limitado a causa de los costos. Y es en este sentido donde este trabajo presenta su mayor ventaja.

Una herramienta de *software* como la lograda permite, de una manera simple y casi transparente para los usuarios sin conocimiento profundo de geomática, publicar la información geográfica que disponen. Y sin costos de licencia, puesto que toda la plataforma necesaria es de código abierto y gratuito. Esto último lo hace bastante atractivo para organismos de gobierno regional y comunal.

La extensión del *framework* de desarrollo *web* CodeIgniter, con las librerías para el manejo de objetos espaciales, aunque menos visible desde el punto de vista de usuarios finales, también resulta ser un aporte. Permitirá contar con una base para ofrecer desarrollo de aplicaciones con características GIS de dominio específico, a muy bajo costo (comparado con el desarrollo utilizando las tecnologías de ESRI).

Ambos desarrollos (librerías y aplicación administradora), constituyen un producto valioso, y altamente rentable, para empresas que ofrecen servicios de implementación y desarrollo de *software* GIS.

[1]Entendido aquí en el amplio sentido del término: profesionales capacitados, información estructurada, procedimientos, *hardware* y herramientas de *software*.

5.2. Trabajo futuro, pendientes.

Evidentemente el producto logrado en este trabajo es perfectible. Tanto para las extensiones a CodeIgniter, como para la aplicación administradora, será el uso el que dictará el camino que tome el desarrollo. Sin embargo, esto no impide establecer al menos el sentido del camino que debe tomar el desarrollo, con algunas recomendaciones.

Comparando la idea que se tenía al comienzo del desarrollo con los resultados obtenidos, se pueden listar las siguientes tareas pendientes que quedaron fuera del alcance de este trabajo por diversas razones (principalmente tiempo):

- Generador de archivos *mapfile*: sería de gran utilidad que la aplicación de administración incluyera un generador de archivos *mapfile*. Una herramienta tal permitiría a los usuarios combinar las capas de información geográfica de distintas fuentes en un documento de mapas de manera simple y rápida, y sin necesidad de tener conocimientos en la gramática y sintaxis de los *mapfiles*.

- Soporte para imágenes raster en la *geodatabase*: actualmente existe el deseo de permitir el almacenamiento eficiente de imágenes georreferenciadas en PotgreSQL. Aunque fuera del alcance de este trabajo, una extensión de PostGIS para que se puedan almacenar raster en la base de datos resulta bastante atractivo.

- Aplicaciones Móviles: la integración con dispositivos móviles resulta de gran interés y utilidad. En este sentido, un camino a tomar es habilitar a la aplicación administradora para que genere aplicaciones para dispositivos móviles.

- Manejo de geometrías en 3D: aunque el interés de las Aplicaciones Web de Mapas no es convertirse en un visualizador 3D, resultaría de utilidad (y no es muy costoso además) contar con soporte para objetos con 3 dimensiones en las librerías espaciales.

El interés de las empresas que implementan GIS en organizaciones pequeñas y medianas, es contar con una herramienta simple, flexible y de bajo costo, que implemente en una escala menor, la mayor cantidad de funcionalidades que las herramientas comerciales más populares.

Referencias

[1] M. F. Goodchild, "What is Geographic Information Science?, *NCGIA Core Curriculum in GIScience*." Internet `http://www.ncgia.ucsb.edu/giscc/units/u002/u002.html`, 1997. [Descargado 6-septiembre-2009].

[2] R. Tomlinson, *Thinking About GIS: Geographic Information System Planning for Managers*. Esri Press, 2005.

[3] A. Moreno, *Sistemas y Análisis de la Información Geográfica*. RaMa, 2005.

[4] L. Tao, L. Meng, J. Fang, J. Li, Z. Chen, and C. Deqin, "Research and Realization of Web GIS framework Based on XML," in *International Workshop on Geoscience and Remote Sensing*, pp. 69–72, IEEE, 2008.

[5] Stanford University Libraries and Academic Information Resources, "Featured GIS Websites." Internet `http://www-sul.stanford.edu/depts/gis/siteofweek.html`, 2009. [Descargado 6-septiembre-2009].

[6] D. Magazine, "Web Map Gallery." Internet `http://www.directionsmag.com/webmapgallery`, 2009. [Descargado 6-septiembre-2009].

[7] G. Euskadi, "Portal Geo Euskadi." Internet `http://www.geo.euskadi.net`, 2009. [Descargado 6-septiembre-2009].

[8] ESRI, "Web ADF overview." Internet `http://edndoc.esri.com/arcobjects/9.2/NET_Server_Doc/developer/ADF/adf_overview.htm`, 2009. [Descargado 6-septiembre-2009].

[9] ESRI, "What is a Geodatabase?." Internet http://help.arcgis.com/en/arcgisserver/10.0/help/arcgis_server_dotnet_help/0093/0093000000q9000000.htm, 2010. [Descargado 10-junio-2010].

[10] C. Portele, "OpenGIS Geography Markup Language (GML) Encoding Standard (Open GIS Standard OGC 07-036)," tech. rep., Open Geospatial Consortium Inc., August 2007.

[11] M. Innerebner, M. Böhlen, and I. Timko, "A Web-enabled extension of a spatio-temporal DBMS," in *GIS '07: Proceedings of the 15th annual ACM International Symposium on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 1–8, ACM, 2007.

[12] C.-H. Huang, T.-R. Chuang, D.-P. Deng, and H.-M. Lee, "Efficient GML-native Processors for Web-Based GIS: Techniques and Tools," in *GIS '06: Proceedings of the 14th annual ACM International Symposium on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 91–98, ACM, 2006.

[13] T. Wilson, "OGC Implementation Specification 07-147r2: OGC KML," tech. rep., Open Geospatial Consortium Inc., April 2008.

[14] Google Code, "KML Reference." Internet http://code.google.com/apis/kml/documentation/kmlreference.html, 2010. [Descargado 10-junio-2010].

[15] Open GIS Consortium, "OpenGIS Simple Features Specification for SQL, Revision 1.1," 1999.

[16] J. Herring, "OpenGIS Implementation Specification for Geographic Information - Simple Feature Access," 2006.

[17] C. Convis, "The Nature of Geographic Information Systems." Internet http://www.conservationgis.org/gishistory/gishistry2.html, 2010. [Descargado 10-junio-2010].

[18] A. Cate, Jr., D. Semmens, D. P. Guertin, and D. C. Goodrich, "DOTAGWA: A Case Study in Web-Based Architectures for Connecting Surface Water Models to Spatially

Enabled Web Applications," in *SCSC: Proceedings of the 2007 Summer Computer Simulation Conference*, (San Diego, CA, USA), pp. 885–892, Society for Computer Simulation International, 2007.

[19] S. Wang, "GISolve Toolkit: Advancing GIS through Cyberinfrastructure," in *GIS '08: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 1–2, ACM, 2008.

[20] ESRI, "ArcGIS Server in Practice Serier." White paper (`http://www.esri.com/library/whitepapers/pdfs/arcgis-server-high-capacity.pdf`), 2009. [Descargado 20-noviembre-2009].

[21] ESRI, "ArcGIS Server 9.3.1 Functionality Matrix." White paper (`http://www.esri.com/library/brochures/pdfs/arcgis92-functionality-matrix-list.pdf`), 2009. [Descargado 20-noviembre-2009].

[22] ESRI, "Who uses ArcGIS Server." White paper (`http://www.esri.com/library/whitepapers/pdfs/arcgis-server.pdf`), 2009. [Descargado 20-noviembre-2009].

[23] S. Freitas, M. B. Carmo, and A. P. Afonso, "A Personalized Visualization Tool for Geo-referenced Information," in *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, (New York, NY, USA), pp. 398–398, ACM, 2005.

[24] R. R. Vatsavai, T. E. Burk, B. T. Wilson, and S. Shekhar, "A Web-based Browsing and Spatial Analysis System for Regional Natural Resource Analysis and Mapping," in *GIS '00: Proceedings of the 8th ACM International Symposium on Advances in Geographic Information Systems*, (New York, NY, USA), pp. 95–101, ACM, 2000.

[25] M. Careem, C. De Silva, R. De Silva, L. Raschid, and S. Weerawarana, "Demonstration of Sahana: Free and Open Source Disaster Management," in *Proceedings of the 8th Annual International Conference on Digital Government Research*, pp. 266–267, Digital Government Society of North America, 2007.

[26] R. F. Chavez, "Generating and Reintegrating Geospatial Data," in *DL '00: Proceedings of the 5th ACM Conference on Digital Libraries*, (New York, NY, USA), pp. 250–251, ACM, 2000.

[27] R. Ellis, "CodeIgniter User Guide." Internet `http://codeigniter.com/user_guide/`, 1997. [Descargado 20-noviembre-2009].

Apéndices

A . Requerimientos aplicación administradora.

RS0002 - Aplicación de Administración	
Descripción	El sistema debe proveer de una aplicación de administración, en ambiente web.
Tipo	Funcional
T. Usuario Asociado	Implementador
RS0003 - Crear AWM	
Descripción	La aplicación de administración debe permitir crear, en una sucesión de pasos simples de configuración, una Aplicación Web de Mapas.
Tipo	Funcional
T. Usuario Asociado	Implementador
RS0004 - Edición de AWM	
Descripción	La aplicación de administración debe permitir gestionar y editar las Aplicaciones Web de Mapas existentes.
Tipo	Funcional
T. Usuario Asociado	Implementador
RS0005 - Eliminación de AWM	
Descripción	La aplicación de administración debe permitir eliminar Aplicaciones Web de Mapas del servidor.
Tipo	Funcional
T. Usuario Asociado	Implementador
RS0012 - Extensión de AWM	
Descripción	El sistema debe permitir, mediante librerías de funciones ad-hoc y un mecanismo de extensión de funcionalidades, crear Aplicaciones Web de Mapas de acuerdo a dominios específicos, a partir de la plantilla base.
Tipo	Funcional
T. Usuario Asociado	Desarrollador

RS0015 - Estándares OGC para BD	
Descripción	El sistema debe seguir los estándares del OGC respecto del tratamiento de la información almacenada en bases de datos espaciales.
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0016 - Trabajo con datos vectoriales	
Descripción	El sistema debe proveer de librerías para el trabajo con datos en archivos con información vectorial (shapefiles, kml, etc.).
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0017 - Trabajo con imágenes raster	
Descripción	El sistema debe proveer de librerías para el trabajo con imágenes raster.
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0018 - Trabajo con WMS	
Descripción	El sistema debe proveer de funciones para el trabajo con datos surtidos por Web Map Services.
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0019 - Uso de GML	
Descripción	El sistema debe trabajar con datos en GML.
Tipo	Funcional
T. Usuario Asociado	Desarrollador

RS0021 - Extensión temporal	
Descripción	El sistema debe proveer de funciones destinadas a la creación de Aplicaciones Web de Mapas dinámicas en el tiempo, esto es, que representan objetos cuya posición varía en el tiempo.
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0022 - Restricciones a los datos	
Descripción	El sistema debe proveer de funciones para manejar restricciones topológicas sobre los datos espaciales.
Tipo	Funcional
T. Usuario Asociado	Desarrollador
RS0001 - Instalación	
Descripción	La instalación del sistema debe ser simple, y requerir las mínimas configuraciones para comenzar a operar.
Tipo	Usabilidad
T. Usuario Asociado	Implementador
RS0006 - Administración 2.0	
Descripción	La aplicación de administración debe ser simple y amigable en su uso. En particular, debe asemejarse a las aplicaciones web populares como Google, Facebook en su diseño y simpleza.
Tipo	Usabilidad
T. Usuario Asociado	Implementador
RS0011 - Portabilidad	
Descripción	El sistema no debe agregar complejidades significativas al entorno necesario por el framework base (CodeIgniter), de modo de ser tan portable como este.
Tipo	Portabilidad
T. Usuario Asociado	Desarrollador,Implementador

RS0014 - Estándares OGC para intercambio	
Descripción	El sistema debe implementar los estándares del OGC para el intercambio, representación y transformación de datos espaciales, brindando librerías que puedan trabajar de acuerdo a estos estándares.
Tipo	Interoperabilidad
T. Usuario Asociado	Desarrollador
RS0007 - Framework base	
Descripción	El sistema debe extender el framework PHP CodeIgniter.
Tipo	Interfaz
T. Usuario Asociado	Desarrollador,Implementador
RS0008 - Gestión de Mapas	
Descripción	El sistema debe utilizar, para la generación y trabajo con mapas web, el servidor de mapas MapServer y las librerías PHP MapScript.
Tipo	Interfaz
T. Usuario Asociado	Desarrollador,Implementador
RS0009 - Bases de Datos	
Descripción	El sistema debe soportar como mínimo el trabajo con datos en PostgreSQL, extendido con PostGIS como motor de datos espaciales.
Tipo	Interfaz
T. Usuario Asociado	Desarrollador,Implementador
RS0010 - Servidor Web	
Descripción	El sistema, como aplicación web, debe ejecutarse en un servidor Apache.
Tipo	Interfaz
T. Usuario Asociado	Desarrollador,Implementador
RS0013 - Documentación	
Descripción	El sistema, como extensión de CodeIgniter, debe seguir las normas de documentación de este, y documentarse el 100 % de las funciones agregadas.
Tipo	Documentacion
T. Usuario Asociado	Desarrollador

B . Algoritmo de Vincenty para distancia elipsoidal.

Consideremos las siguientes definiciones:

a, b semiejes mayor y menor del elipsoide de referencia

$f = \frac{(a-b)}{a}$	achatamiento del elipsoide
ϕ_1, ϕ_2	latitudes de los puntos
L	diferencia entre las longitudes de los puntos
$U_i = atan((1-f)tan(\phi_i))$	latitudes reducidas, $i \in \{1,2\}$

$\lambda = L$, $\lambda_l = 2\pi$

El algoritmo (pseudocódigo) es el siguiente:

Itero mientras $|\lambda - \lambda_0| \geq \epsilon$

$ssig \leftarrow \sqrt{((cos(U_2)sin(\lambda))^2 + (cos(U_1)sin(U_2) - sin(U_1)cos(U_2)cos(\lambda))^2)}$

$csig \leftarrow sin(U_1)sin(U_2) + cos(U_1)cos(U_2)cos(\lambda)$

$sig \leftarrow atan(\frac{ssig}{csig})$

$salp \leftarrow \frac{cos(U_1)cos(U_2)sin(\lambda)}{ssig}$

$c2alp \leftarrow 1 - salp^2$

$c2sm \leftarrow csig - \frac{2sin(U_1)sin(U_2)}{c2alp}$

$C \leftarrow \frac{f}{16c2alp(4+f(4-3c2alp))}$

$\lambda_0 \leftarrow \lambda$

$\lambda \leftarrow L + (1+C)fsalp\{sig + Csalp(c2sm + Ccsig(2c2sm^2 - 1))\}$

$u2 \leftarrow c2alp\frac{a^2-b^2}{b^2}$

$A \leftarrow \frac{1+u2}{16384}\{4096 + u2(-768 + u2(320 - 175u2))\}$

$B \leftarrow \frac{u2}{1024}\{256 + u2(-128 + u2(74 - 47u2))\}$

$dsig \leftarrow Bssig\{c2sm + \frac{B}{4}(csig(-1 + 2c2sm^2) - \frac{B}{6}c2sm(-3 + 4ssig^2)(-3 + 4c2sm^2))\}$

$s \leftarrow BA(sig - dsig)$

$alp1 \leftarrow atan(\frac{cos(U_2)sin(\lambda)}{cos(U_1)sin(U_2) - sin(U_1)cos(U_2)cos(\lambda)})$

$alp2 \leftarrow atan(\frac{cos(U_1)sin(\lambda)}{-sin(U_1)cos(U_2) + cos(U_1)sin(U_2)cos(\lambda)})$

Donde s es la distancia elipsoidal entre los puntos, y $alp1, alp2$ son los azimuts inicial y final del arco sobre el elipsoide.