
INDICE

INDICE	i
INDICE DE FIGURAS	vi
INDICE DE TABLAS	ix
INDICE DE ANEXOS	xiii
RESUMEN	xv
SUMMARY	xvi
1. INTRODUCCION	1
2. OBJETIVOS	5
2.1. Objetivo General	6
3. ANALISIS DEL PROBLEMA	7
3.1. Definición del Problema	8
4. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)	9
4.1. Introducción	10
4.2. Descripción	10
4.3. Información geográfica distribuida (IGD)	11
4.4. Shapefile	11
4.4.1. Tipos de Shapes o Geometrías	12

4.5. OpenGis	14
4.5.1. Postgresql	15
4.5.2. PostGis.....	15
4.5.3. Esquema conceptual.....	16
4.5.4. Definición de tablas básicas.....	17
4.5.5. Crear una tabla espacial.....	19
4.5.6. Ingresar datos espaciales.....	21
5. JAVA.....	22
5.1. Recursos necesarios.....	23
5.2. Applet y Aplicaciones	25
5.2.1. Applet.....	25
5.2.1.1 Pruebas	25
5.2.2. Aplicaciones	28
5.2.2.1. Pruebas.....	28
5.3. JSP y Servlet.....	29
5.3.1. Pruebas	30
5.3.2. Comunicación entre Applet y Servlet	31
5.4. Archivos JAR.....	33
5.4.1. ¿Cómo crear un archivo JAR?	34
5.4.2. Firmar digitalmente un JAR.....	35
5.5. Conexión a Base de Datos – JDBC	36
5.5.1. Introducción.....	36
5.5.2. Conceptos básicos	36

5.5.3. Conexión a Postgresql	38
5.6. Java Web Start - JNLP	39
5.6.1. Pruebas	41
5.6.2. Descripción.....	42
5.6.3. Tipo MIME	43
5.7. Gráficas.....	43
5.7.1. ¿Cómo usar Api 2D?.....	45
5.8. Múltiples Hilos y creación de eventos	45
5.8.1. Múltiples Hilos	46
5.8.2. Creación de eventos	48
6. GEOTOOLS	52
6.1. Introducción.....	53
6.2. Geotools 1 (GT1).....	53
6.3. Geotools 2 (GT2).....	54
6.4. ¿Cómo utilizar Geotools?.....	56
6.4.1. Tutorial de Geotools 1	56
6.4.2. Tutorial de Geotools 2	61
7. VISOR DE DATOS GEOGRÁFICOS.....	64
7.1. Introducción.....	65
7.2. Etapas de la investigación.....	65
7.3. Análisis y especificación de requerimientos	69
7.3.1. Requisitos Funcionales	69

7.3.2. Requisitos no Funcionales	70
7.3.2.1. Documentación	70
7.3.2.2. Consideraciones de Hardware	70
7.3.2.3. Ambiente Físico.....	70
7.3.3. Modelos del Sistema	71
7.3.3.1. Componentes del Modelo de Casos de Uso	71
7.3.3.2. Modelo de casos de uso.....	73
7.3.3.3. Descripción de casos de uso.....	73
7.4. Análisis.....	77
7.4.1. Modelo de generalización.....	77
7.4.2. Descripción general del modelo de clases de análisis	78
7.4.2.1. Interfaces.....	78
7.4.2.2. Procesos	78
7.4.2.3. Entidades	79
7.4.3. Modelo de análisis de la aplicación	81
7.5. Diseño	82
7.5.1. Diseño de interfaces.....	82
7.5.2. Diagrama de clases.....	86
7.6. Implementación	87
La descripción de la implementación consta de lo siguiente:.....	87
7.6.1. Detalle de clases	87
7.7. Prueba.....	102

8. DESARROLLOS FUTUROS	104
8.1. Comentarios.....	105
9. CONCLUSIÓN Y COMENTARIOS	107
9.1. Conclusión	108
10. BIBLIOGRAFÍA	109
ANEXOS	110

INDICE DE FIGURAS

4. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

4.5.3. Esquema conceptual OpenGis.....	16
4.5.5. Vista de la tabla geometry_columns después de crear tablas geométricas.....	21

5. JAVA

5.2.1.1.1. Vista en el navegador de del ejemplo Applet anterior	27
5.4.1. Estructura básica del archivo manifest.....	35
5.5.2.1. Conexión JDBC.....	37
5.5.2.2. Conexión JDBC con ODBC.....	37
5.6. Proceso de descarga manejado por Java Web Start	40
5.7. Gráficos contruidos con Api 2D	45

6. GEOTOOLS

6.4.1. Demo Geotools 1	60
6.4.2. Demo Geotools 2	63

7. VISOR DE DATOS GEOGRÁFICOS

7.3.3.1.1. Representación de un actor.	71
7.3.3.1.2. Representación de un caso de uso.	71
7.3.3.1.3. Representación gráfica de una relación.	72

7.3.3.2. Modelo de casos de uso	73
7.4.1. Modelo de generalización	77
7.4.3. Modelo de análisis de la aplicación	81
7.5.1.1. Elementos de interfaces	82
7.5.1.2. Pantalla inicial	83
7.5.1.3. Pantalla detalles	84
7.5.1.4. Pantalla progreso	85
7.5.1.5. Pantalla ayuda.....	85
7.5.2. Diagrama de clases.....	86
7.6.1.1. Clase Visor	87
7.6.1.2. Formulario de la clase Visor	88
7.6.1.3. Clase vtnMapas.....	89
7.6.1.4. Formulario de la clase vtnMapas.....	90
7.6.1.5. Clase CollectionMaps.....	91
7.6.1.6. Clase ProgressDialog	93
7.6.1.7. Formulario de la clase ProgressDialog	93
7.6.1.7. Clase LoaderFile	94
7.6.1.8. Clase Loaders	96
7.6.1.9. Clase MapPane	96
7.6.1.10. Clase vntDetalles	98
7.6.1.11. Formulario de la clase vntDetalles	98
7.6.1.12. Clase vntDetalleTabla	99
7.6.1.13. Formulario de la clase vntDetallesTabla	101
7.7. Visor de mapas geográficos.....	103

ANEXO C. Tomcat 4.0

c.4. Directorios de trabajo de tomcat..... 140

ANEXO D. Compilación GT2

d.3. Estructura de directorios de geotools2..... 150

ANEXO F. Código del Visor de datos geográficos

f.1. Diagrama de clase 162

INDICE DE TABLAS

4. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

4.4.1. Tipos de shapes	13
4.5.4.1. Tabla spatial_ref_sys	17
4.5.4.2. Tabla geometry_columns	18
4.5.4.3. Tabla feature table view	19
4.5.5.1. Tabla calles_geom	20
4.5.5.2. Tabla parques	20
4.5.5.3. Tabla calles	20

5. JAVA

5.5.6. Insertar datos espaciales	21
5.2.1.1.1. Ejemplo de Applet	26
5.2.1.1.2. Página web del ejemplo de Applet	27
5.2.2.1.1. Aplicación en modo texto	28
5.2.2.1.2. Aplicación en modo gráfico	29
5.3.1.1. Ejemplo Jsp.....	30
5.3.1.2. Ejemplo Servlet	30
5.3.2. Comunicación Servlet/Applet	33
5.5.3. Conexión a Postgresql	38
5.6.1. Ejemplo archivo JNLP	41
5.8.1. Ejemplo de multihilo	47

5.8.2.1. Ejemplo de creación de interfaces	49
5.8.2. Ejemplo de creación de eventos dentro de un multihilo	51

6. GEOTOOLS

6.4.1. Estructura del archivo Key	58
---	----

7. VISOR DE DATOS GEOGRÁFICOS

7.3.3.3.1. Caso de uso: Selección.....	74
7.3.3.3.2. Caso de uso: Cargar Mapa.	74
7.3.3.3.3. Caso de uso: Visualiza.....	74
7.3.3.3.4. Caso de uso: Visualiza Capas.....	75
7.3.3.3.5. Caso de uso: Visualiza datos referente al mapa.	75
7.3.3.3.6. Caso de uso: Visualiza características.....	75
7.3.3.3.7. Caso de uso: Imprimir.	76
7.3.3.3.8. Caso de uso: Ayuda.....	76
7.4.2.1. Interfaces	78
7.4.2.2. Procesos	79
7.4.2.3. Entidades	80
7.6.1. Archivo de configuración maps.dat	92

ANEXO A. Postgresql

a.4.1. Lista las funciones.....	118
a.4.2. Listar las vistas.....	118
a.4.3. Listar las tablas	118
a.4.4. Eliminar función.....	119
a.4.5. Listar contenido de la función.....	119
a.4.6. Eliminar vista	119
a.4.7. Listar contenido de la vista	119
a.4.8. Listar usuarios.....	119
a.5. Ejemplo de procedimientos almacenado.....	121
a.6.1. Tabla Stock	121
a.6.2. Procedimientos almacenado que verifica el stock.....	122
a.6.3. Llama al procedimiento de verificación de stock	122

ANEXO B. PostGis

b.2.1.1. Prueba de código PostGis.....	131
b.2.1.2. Prueba de código PostGis, página web	132

ANEXO C. Tomcat 4.0

c.1. Archivo de configuración de servicios de tomcat.....	136
c.4. Configuración mínima de web.xml.....	141
c.5. Archivo de comunicación entre Tomcat y Apache.....	146

ANEXO E. Pruebas con Geotools

- e.1.1. Demo Geotools1 - applet 155
- e.1.2. Demo Geotools1 – web 156
- e.2.1. Demo Geotools2 - JFrame 158
- e.2.1. Demo Geotools2 - Class 160

INDICE DE ANEXOS

a. Postgresql	111
a.1. Levantar los servicios.....	111
a.2. Habilitar el puerto de conexión.....	114
a.3. ¿Como usar Postgresql?.....	116
a.4. Automatización de Procesos Habituales.....	118
a.5. Procedimientos Almacenados - Pl/pgsql.....	119
a.6. Pruebas.....	121
b. PostGis	123
b.1. Instalación y Configuración	123
b.2. Compilar PostGis.jar	126
b.2.1. Pruebas.....	127
c. Tomcat 4.0	133
c.1. Instalación	133
c.2. Descripción del contenido del paquete.....	137
c.3. Configuración	139
c.4. Modo de uso.....	140
c.5. Comunicación entre Tomcat y Apache.....	141

d. Compilación de GT2	148
d.1. Java JAI	148
d.2. Apache Maven	149
d.3. Compilación	150
e. Pruebas con Geotools	152
e.1. GT1	152
e.2. GT2	156
f. Código del Visor de datos geográficos	161
f.1. Código del Visor	161

RESUMEN

El presente trabajo tiene como objeto, desarrollar una aplicación que permita desde un servidor remoto y por medio de Internet, visualizar datos geográficos en formato vectorial, logrando proveer de una interfaz multiplataforma robusta y de bajo costo de desarrollo. Para lograr esto se contempla el estudio de distintas herramientas libres (en su mayoría) que contribuyan a lograr el objetivo, por medio de las cuales se comienza a trazar un camino de investigación que al principio parece un tanto incierto.

Al comenzar, se plasman los conocimientos adquiridos por medios de cortos tutoriales, que en su mayoría pretenden ser una introducción a los distintos temas que encierran un Sistemas de Información Geográfica (SIG).

Posteriormente se realizan algunas pruebas, que confirman y consolidan los conocimientos adquiridos en el proceso de investigación. Luego de las cuales queda de manifiesto las ventajas, desventajas y complejidad de alguna de ellas.

Tras la investigación y evaluación de las herramientas, se comienza la construcción de un prototipo desarrollado sobre una plataforma Java, que a su vez es complementado con las librerías de código abierto proporcionadas por el proyecto Geotools, orientado a desarrollos SIG y que es mantenido por la comunidad informática.

SUMMARY

The present work has like object, to develop an application that allows from a remote servant and by means of Internet, to visualize geographical data in vectorial format, being able to provide multi of an interface robust platform and of low development cost. To achieve this the study of different free tools it is contemplated (in their majority) that contribute to achieve the objective, by means of which you begins to trace an on the way to investigation that at the beginning seems an uncertain point.

When beginning, the knowledge are captured acquired by means of short chapters that seek to be an introduction to the different topics in their majority that they contain a Geographical Information Systems (GIS).

Later on they are carried out some tests that confirm and they consolidate the knowledge acquired in the investigation process. After which it is of manifesto the advantages, disadvantages and complexity of some of them.

After the investigation and evaluation of the tools, you begins the construction of a prototype developed on a platform Java that in turn is supplemented with the bookstores of open code provided by the project Geotools, guided to developments GIS and that it is maintained by the computer community.

1. INTRODUCCION

Un Sistema de Información Geográfica (SIG), está ligado al uso de un conjunto de herramientas, las cuales existen en el mercado, pero a su vez no están al alcance de la gran mayoría. Esto es, por el alto costo del software propietario, imprescindible, si se piensa abordar un desarrollo de tales características. Y es precisamente ahí donde surgen las inquietudes y fundamentos que dan origen al presente trabajo.

Así, el objetivo de este trabajo está orientado a analizar y evaluar las opciones presentes en el mercado para concretar una aplicación que permita al usuario de forma fácil, visualizar datos geográficos en formato vectorial a través de Internet y por medio de un servidor remoto dispuesto previamente.

La plataforma de desarrollo seleccionada es un lenguaje de programación de actualidad, como lo es Java, así como también lo son; el motor de datos Postgresql usado para soportar el estándar de OpenGis y la plataforma Linux Red Hat 9 sobre la cual se encuentra instalado.

Cada uno de los capítulos aporta información relevante para lograr el objetivo propuesto y que a su vez reflejan los conocimientos y vivencias que se fueron dando conforme el proceso de investigación tomaba distintos rumbos. Los que proporcionaron la experiencia suficiente para concluir en una aplicación final.

El capítulo N° 2 se habla de los objetivos de este trabajo. En el capítulo N° 3 se presentan los fundamentos que dan origen a este trabajo. En el capítulo N° 4 se detalla el manejo del motor de base de datos Postgresql y la herramienta PostGis, que agrega nuevas funcionalidades orientadas al manejo de datos geométricos.

El capítulo N° 5 pretende introducir a los dos tipos de almacenamientos de datos geométricos a los cuales se hace referencia en los distintos capítulos.

El capítulo N° 6 se enfoca sobre los ámbitos mas relevantes sobre el lenguaje de programación Java, los cuales son usados en la construcción del “Visor de Datos Geográficos”.

El capítulo N° 7 detalla alguna de las cualidades que poseen las dos versiones disponibles actualmente de Geotools, que es un conjunto de librerías para Java orientadas al desarrollo de aplicaciones SIG.

El capítulo N° 8 se enfoca en describir las etapas del desarrollo del visor de datos geográficos, a partir de los requerimientos hasta la implementación y prueba.

En el capítulo N° 9 se comentan los desarrollos futuros pendientes para ampliar y mejorar el prototipo inicial propuesto. En el capítulo N° 10 se establece una conclusión al problema.

Finalmente en el anexo A se detalla los pasos a seguir para configurar Postgresql sobre un servidor Linux.

En el anexo B se indican los pasos para montar PostGis sobre Postgresql, desde la compilación de los módulos hasta su configuración.

En el anexo C se encuentra la información necesaria para instalar y configurar el servidor de servlet Tomcat que se usa en algunas pruebas de comunicación entre Applet y Servlet, con el fin de encontrar un método alternativo de conexión a bases de datos.

En el anexo D se especifica el proceso de compilación de Geotools2 (GT2) .

En el anexo E se presenta el código fuente de las pruebas realizadas en el capítulo N° 7.4 sobre Geotools.

En el anexo F se presenta el código fuente de algunas de las clases más importantes del “Visor de datos geográficos”.

2. OBJETIVOS

2.1. Objetivo General

Desarrollar un visualizador que permita el despliegue de datos geográficos en formato vectorial, con objeto de ser usado sobre Internet.

Las etapas de este trabajo están enmarcadas por lo siguientes puntos:

- Investigación sobre las herramientas a utilizar, instalación, configuración y utilización.
- Evaluación de las herramientas disponibles y posibles alternativas según funcionalidad y desempeño observado hasta el momento.
- Diseño, implementación y evaluación de una serie de pruebas que permitan adquirir conocimiento sobre la programación gráfica y la comunicación entre los distintos componentes.
- Desarrollo final, que reúne los conocimientos anteriormente adquiridos y las herramientas seleccionadas en un prototipo final.

Mediante el cumplimiento de estas etapas se pretende adquirir los conocimientos necesarios para el logro de los objetivos.

3. ANALISIS DEL PROBLEMA

3.1. Definición del Problema

El tema de este trabajo nace inicialmente como una inquietud basada en dos trabajos de características similares realizados en la Universidad de las Américas, Puebla, México.

- Exportación de datos usando OpenGis por Felix Francisco García Ontiveros.
- Desarrollo de un exportador de datos geográficos en una arquitectura de componentes SIG por Germán Escobar Alonso.

Tomando en cuenta lo antecedentes anteriores, se pretende aportar y ampliar lo expuesto mediante el diseño de una aplicación SIG orientada a usuarios de Internet.

El problema específico consiste en investigar y evaluar las opciones posibles de implementación, que permitan concluir en un “Visor” que obtenga los recursos (mapas) desde una biblioteca remota mediante Internet.

Gracias a las herramientas existentes de código abierto hoy en día, pensar en una solución de este tipo es mucho más factible, por razones económicas.

Éstas proveen de un conjunto de métodos que permiten la manipulación de los datos geográficos como insertar, eliminar, modificar y visualizar los datos espaciales. Estos datos pueden ser almacenados en archivos o bases de datos.

4. SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

4.1. Introducción

Los sistemas de información geográfica (SIG) son hoy en día una herramienta ya de uso masivo. Existe una gran cantidad de software orientado a facilitar el manejo de los datos vectoriales sobre representaciones digitales, aunque el costo de estos es bastante elevado.

Por otra parte, también es posible encontrar librerías y aplicaciones orientadas al manejo de datos geográficos de forma gratuita. Cada vez son más los recursos de los cuales se dispone para emprender un desarrollo de estas características. Sobre la plataforma Java se desarrollan librerías libres cada vez mejores, conforme el lenguaje crece. Geotools es uno de estos proyectos (ver capítulo N° 6).

4.2. Descripción

El objetivo primordial que se espera de un SIG, es que nos permita almacenar, recuperar, analizar y desplegar información geográfica, es decir, disponen de métodos y propiedades destinadas a facilitar el manejo de este tipo de información.

Se pueden dividir en tres partes:

- Herramientas para la entrada y manipulación de información geográfica.
- Un sistema de administración de base de datos (DBMS).
- Herramientas que soportan consultas, análisis y visualización de elementos geográficos

4.3. Información geográfica distribuida (IGD)

Es la utilización de herramientas o proyectos SIG de forma remota a través de Internet, lo que permite compartir los datos comunes con el resto de los usuarios.

4.4. Shapefile

La empresa ESRI propietaria de ArcView (software de administración y visualización geográfica) crea el formato Shapefile, que es un conjunto de archivos que contienen las geometrías y sus datos asociados.

Tres son los tipos de archivos que forman parte de este formato:

- Shapefile (*.shp): Almacena las geometrías.
- DBase (*.dbf): Almacena los atributos descriptivos.

-
- Index (*.shx): Archivo de índice. La correspondencia entre los registros de un dbf y un shp es de uno es a uno.

El dbf y el shx deben tener el mismo nombre que tiene el shp para ser reconocidos como parte de él.

4.4.1. Tipos de Shapes o Geometrías

Los shapes son almacenados en los shapefiles y pueden ser de los siguientes tipos:

Valor	Shape Tipo
0	Shape Nulo
1	Punto
3	Polilínea
5	Polígono
8	Multipunto
11	PuntoZ
13	PoliLineaZ
15	PolígonoZ
18	MultiPuntoZ
21	PuntoM
23	PoliLineaM

25	PolígonoM
28	MultiPuntoM
31	MultiPatch

(Tabla 4.4.1) Tipos de shapes

Los números que no se encuentran en la lista están reservados para desarrollos futuros. Los shapefiles están limitados a almacenar sólo un tipo de geometría.

Un mapa esta representado por un gran conjunto de información vectorial, que se organiza por medio de capas (coberturas). Cada capa esta formada por geometrías de un mismo tipo, como por ejemplo polígonos (ver tabla 4.4.1) y cada una de éstas capas se almacenan en un Shapefile.

Además, con motivo de mantener un orden conceptual sobre las geometrías, las capas suelen subdividirse según lo que estas representan, es decir, todas las geometrías que representan por ejemplo las calles dentro del mapa, se almacenan en un Shapefile.

De la misma forma, todas las geometrías que representan ríos se almacenan en otro Shapefile. Pudiendo visualizarse una u otra cobertura seleccionando sólo el archivo correspondiente.

4.5. OpenGis

Formato estándar sobre SQL con soporte de geometrías, orientada al uso remoto mediante servidores de base de datos relacionales.

OpenGis fue creado con la sola idea de unificar los formatos propietarios existentes proveyendo de una especificación libre para SQL con soporte espacial.

Las empresas que participaron en dicha construcción son:

- Environmental Systems Research Intitute, Inc. (ESRI)
- IBM Corporation.
- Informix Software, Inc.
- MapInfo Corporartion.

La gran mayoría de los motores de datos no incluyen soporte para este tipo operaciones en sus distribuciones, pero existen utilidades que se montan sobre el servidor y que permite ampliar los tipos de datos básicos a tipos geométricos, además de proporcionar la estructura de tablas para el almacenamiento y algunos procedimientos almacenados para el manejo de tales acciones.

Dado que el motor de datos soporta datos espaciales y datos no espaciales pueden utilizarse ambos tipos conjuntamente. OpenGis esta orientado al procesamiento de datos en un ambiente de redes a diferencia del formato

Shapefile que por ser un archivo plano se relaciona con el procesamiento de datos monousuario.

4.5.1. Postgresql

Muchos son los motores de datos existentes en el mercado, Postgresql se destaca por sus capacidades desde hace un tiempo y cada vez toma mayor fuerza. OpenGis es soportado por este motor de datos por medio de PostGis, que es una herramienta que le permite manipular geometrías.

La manipulación y administración de Postgresql se realiza por medio de la consola o terminal (línea de comando) de Linux, aunque existen herramientas gráficas y de administración remota que realizan las mismas acciones.

En el anexo A se especifican los detalles de instalación, configuración y utilización de este motor de datos.

4.5.2. PostGis

Esta herramienta se monta sobre el servidor de datos Postgresql y nos permite manejar nuevos tipos de datos que no son parte del motor.

Existen versiones de Postgresql para Linux y Windows, al igual existen versiones de PostGis para ambos sistemas operativos.

Ejemplos de la representación en modo texto de los tipos de datos geométricos

que agrega PostGis a Postgresql:

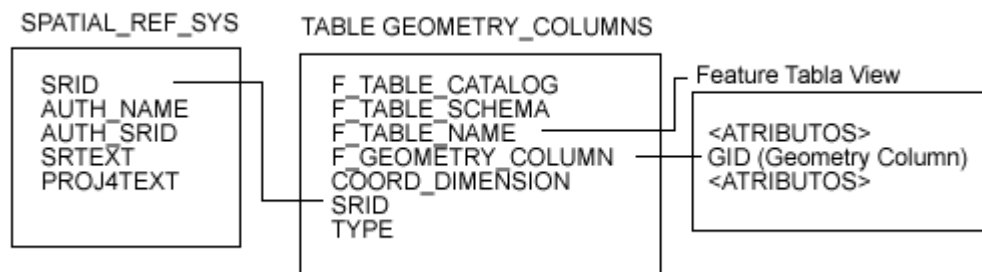
```
POINT(0 0 0)
LINESTRING(0 0,1 1,1 2)
POLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0)),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
MULTIPOINT(0 0 0,1 2 1)
MULTILINESTRING(((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0)),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1
-1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
GEOMETRYCOLLECTION(POINT(2 3 9),LINESTRING((2 3 4,3 4 5))
```

El detalle de la estructura de PostGis se encuentra en el sitio oficial:

<http://postgis.refrations.net/>

El proceso de instalación y compilación de PostGis sobre Postgresql además de las pruebas se detalla en el anexo B.

4.5.3. Esquema conceptual



(Figura 4.5.3) Esquema conceptual OpenGis

4.5.4. Definición de tablas básicas

A partir de este punto todo lo referente a OpenGis (creación de tablas, ejemplos, etc) esta basado en el soporte de PostGis para Postgresql.

- SPATIAL_REF_SYS

Almacena datos referentes a las tablas geométricas definidas por el usuario y que se enlazan por medio de SRID.

```
create table spatial_ref_sys(  
srid integer not null primary key,  
auth_name varchar(256),  
auth_srid integer,  
srttext varchar(2048),  
proj4text varchar(2048)  
)
```

(Tabla 4.5.4.1) Tabla spatial_ref_sys

SRID: Valor entero que identifica el sistema de referencia espacial.

AUTH_NAME: El nombre del estándar para el sistema de referencia. Por ejemplo: EPSG.

AUTH_SRID: El identificador según el estándar AUTH_NAME. En el ejemplo anterior es el id según EPSG.

SRTEXT: Una *Well-know text* representación para el sistema de referencia espacial. Ejemplo: WKT para SRS.

- GEOMETRY_COLUMNS

Registra las tablas geométricas definidas por el usuario indicando el tipo y dimensión (2D, 3D).

```
create table geometry_columns(  
  f_table_catalog varchar(256) not null,  
  f_table_schema varchar(256) not null,  
  f_table_name varchar(256) not null,  
  f_geometry_column varchar(256) not null,  
  coord_dimension integer not null,  
  srid integer not null,  
  type varchar(30) not null  
)
```

(Tabla 4.5.4.2) Tabla geometry_columns

F_TABLE_CATALOG,F_TABLE_SCHEMA,F_TABLE_NAME: Distingue totalmente la tabla de características que contiene la columna geométrica.

F_GEOMETRY_COLUMN: Nombre de la columna geométrica en la tabla de características.

COORD_DIMENSION: Dimensión espacial de la columna(2D o 3D).

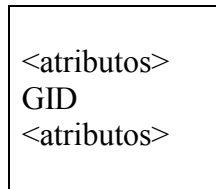
SRID: Es una clave foránea que referencia SPATIAL_REF_SYS.

TYPE: Tipo de objeto espacial. POINT, LINESTRING, POLYGON,

MULTYPOINT, GEOMETRYCOLLECTION. Para un tipo heterogéneo se debe usar el tipo GEOMETRY.

-
- Feature table view

Tabla creada por el usuario, y es definitivamente la encargada de almacenar las geometrías.



(Tabla 4.5.4.3) Tabla feature table view

<atributos>: Son campos definidos por el usuario.

GID: Almacena los datos geométricos.

4.5.5. Crear una tabla espacial

Para crear una tabla con datos espaciales se realizan dos pasos:

- Crear una tabla no espacial (tabla común).
- Luego se añade una columna (campo) espacial a la tabla usando la función *AddGeometryColumn* de PostGis.

AddGeometryColumn(<db_name>, <table_name>, <column_name>, <srid>, <type>, <dimension>)

Ejemplos: Usando una base de datos llamada gdat con soporte para PostGis.

Agrega a la tabla calles_geom el campo 'geom' de tipo 'LINESTRING'.

```
CREATE TABLE CALLES_GEOM (ID int4,NAME varchar(25));  
SELECT AddGeometryColumn( 'gdat', 'calles_geom', 'geom', 423, 'LINESTRING', 2);
```

(Tabla 4.5.5.1) Tabla calles_geom

Agrega a la tabla parques el campo 'parque_geom' del tipo MULTIPOLYGON.

```
CREATE TABLE PARQUES (PARQUE_ID int4,PARQUE_NOMBRE  
varchar(128),PARQUE_FECHA date);  
SELECT AddGeometryColumn( 'gdat' , 'parques' , 'parque_geom' , 128 ,  
'MULTIPOLYGON' , 2 ) ;
```

(Tabla 4.5.5.2) Tabla parques

Agrega a la tabla calles el campo 'calles_geom' del tipo GEOMETRY.

```
CREATE TABLE CALLES(CALLE_ID int4,CALLE_NOMBRE varchar(128));  
SELECT AddGeometryColumn( 'gdat', 'calles', 'calles_geom', -1, 'GEOMETRY', 3);
```

(Tabla 4.5.5.3) Tabla calles

Cada una de las acciones anteriores se registran en la tabla `geometry_columns`:

```
select * from geometry_columns;
```

f_table_catalog	f_table_schema	f_table_name	f_geometry_column	coord_dimension	srid	Type
	gdat	parques	parque_geom	2	128	MULTIPOLYGON
	gdat	calles_geom	geom	2	423	LINestring
	gdat	calles	calles_geom	3	-1	GEOMETRY

(Figura 4.5.5) Vista de la tabla `geometry_columns` después de crear tablas geométricas

4.5.6. Ingresar datos espaciales

Los datos se insertan de forma normal, o sea por medio de sentencias SQL.

Con la particularidad de que se especifican los datos geométricos según las funciones impuestas por PostGis.

```
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText('LINestring(189141 244158,189265 244817)',-
1),'Geordie Rd');

INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText('LINestring(192783 228138,192612 229814)',-
1),'Paul St');

INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText('LINestring(189412 252431,189631 259122)',-
1),'Graeme Ave');

INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText('LINestring(190131 224148,190871 228134)',-
1),'Phil Tee');
```

(Tabla 5.5.6) Insertar datos espaciales

5.1. Recursos necesarios

Java es un lenguaje interpretado, precompilado de forma que pueda ser usado en distintas plataformas. Para ejecutar cualquier aplicación java se requiere de otra aplicación que traduzca el código precompilado a código entendible por la maquina en la cual se ejecuta, a esto se la llama Java Virtual Machine (JVM). Para cada sistema operativo existe una versión de JVM disponible desde <http://java.com/es/>.

Para desarrollar este tipo de aplicaciones existen 3 tipos de paquetes:

- Java 2 Estándar Edition (J2SE).
- Java 2 Enterprise Edition (J2EE).
- Java 2 Micro Edition (J2ME).

Y se pueden obtener en el sitio oficial <http://java.sun.com/>

EL código java se puede escribir en cualquier procesador de texto, aunque existen varios editores que facilitan de forma sustancial el proceso. Ellos son denominados IDE's (Integrated Developer Edition), es decir, un entorno de desarrollo con opciones destinadas a facilitar la programación.

Algunos de los más usados:

- Eclipse (IBM)

<http://www.eclipse.org>

Licencia: Opensource

Plataformas: Windows, Linux, OSX

- JBuilder (Borland)

<http://www.borland.com/jbuilder/>

Licencia: La versión de Evaluación y la Personal son gratis, las avanzadas Profesional y Enterprise son comerciales.

Plataformas: Windows, Linux, Solaris

- Sun ONE Studio (Sun Microsystems)

<http://www.sun.com/forte/ffj/>

Licencia: Esta basado en el IDE opensource Netbeans pero las versiones existentes son todas comerciales.

Plataformas: Todas con Java Virtual Machine (JVM).

- Netbeans (SunMicrosystem)

<http://www.netbeans.org/>

Licencia: Opensource

Plataformas:Todas con JVM.

5.2. Applet y Aplicaciones

5.2.1. Applet

Es una aplicación java que se ejecuta dentro de un navegador web (con soporte para java). Resulta bastante cómodo para el usuario porque no tiene la necesidad de instalar la aplicación.

El Applet tiene acceso total al servidor del cual proviene, pero no así al computador del usuario. De esta forma JVM protege los datos de aplicaciones maliciosas. Hay ocasiones en las cuales es realmente necesario disponer de acceso al disco del usuario y se logra firmando digitalmente los paquetes y es el mismo usuario el que decide si autoriza o no el acceso total al disco.

5.2.1.1 Pruebas

Muestra un cuadro de texto con un mensaje por defecto.

EjemploApplet.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;

public class EjemploApplet extends JApplet {
    JPanel jPanel1 = new JPanel();
    JTextField jTextField1 = new JTextField();
```

```

//Construir el applet
public EjemploApplet () {
}

//Inicializar el applet
public void init() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

//Inicialización de componentes
private void jbInit() throws Exception {
    this.setSize(new Dimension(336, 72));
    jPanel1.setLayout(null);
    jTextField1.setText("Ejemplo Simple.....");
    jTextField1.setBounds(new Rectangle(25, 18, 290, 31));
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jTextField1, null);
}
}

```

(Tabla 5.2.1.1.1) Ejemplo de Applet

EjemploApplet.html

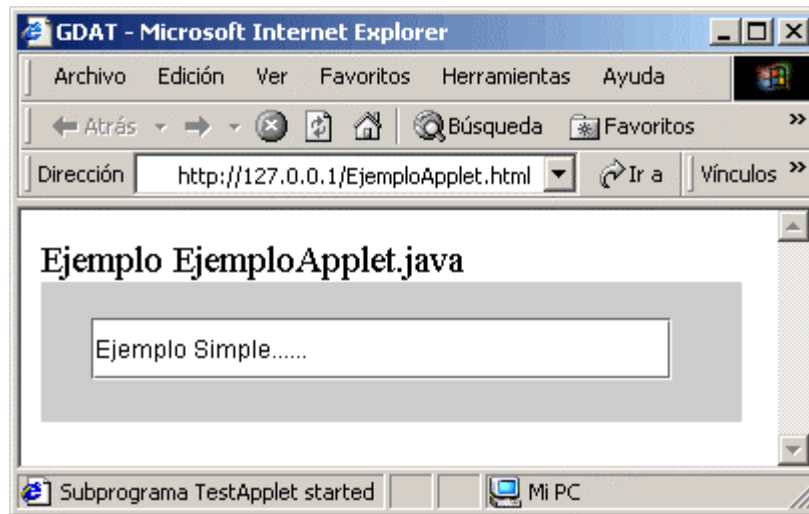
```

<html>
<body>
Ejemplo EjemploApplet.java<br>
<applet
    codebase=""
    code=" EjemploApplet.class"
    name="Test"
    width="350"
    height="70"
    hspace="0"
    vspace="0"
    align="middle"
>
El applet no se puede ejecutar por que no se encuentra el plugin de Java.
</applet>

```

```
</body>  
</html>
```

(Tabla 5.2.1.1.2) Página web del ejemplo de Applet



(Figura 5.2.1.1.1) Vista en el navegador de del ejemplo Applet anterior

5.2.2.Aplicaciones

Se ejecuta directamente sin la necesidad de un navegador web. Como toda aplicación, esta tiene total acceso a los recursos sobre el cual se ejecuta, sin la necesidad de tener que firmar los paquetes.

Existen 2 tipos, los que se ejecutan de forma gráfica (ventanas) y los que se ejecutan en una consola en modo texto.

5.2.2.1. Pruebas

EjemploTexto.java (en modo texto)

```
class EjemploTexto {
    public static void main( String args[] ) {
        System.out.println( "Ejemplo simple..." );
    }
}
```

(Tabla 5.2.2.1.1) Aplicación en modo texto

EjemploGrafico.java (en modo gráfico)

```
import javax.swing.*;
import java.awt.*;

public class EjemploGrafico extends JFrame {
    JTextField jTextField1 = new JTextField();

    public EjemploGrafico () {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public static void main(String[] args) {
    EjemploGrafico app = new EjemploGrafico ();
    app.setSize(400,100);
    app.show();
}

private void jbInit() throws Exception {
    jTextField1.setText("Ejemplo simple...");
    jTextField1.setBounds(new Rectangle(32, 17, 309, 24));
    this.getContentPane().setLayout(null);
    this.getContentPane().add(jTextField1, null);
}
}
```

(Tabla 5.2.2.1.2) Aplicación en modo gráfico

5.3. JSP y Servlet

La construcción de sitios web también es posible mediante Java Server Pages (JSP) y Servlet. Con ambos se obtienen los mismos resultados pero son ligeramente distintos.

El servidor de Servlet y JSP mas usado es Jakarta Tomcat el cual puede adosarse al servidor web Apache y a otros.

Las JSP's son páginas HTML comunes con la particularidad de tener tags java en su interior y no es necesario compilarlo porque el mismo servidor de servlet es el encargado de realizar el proceso. Por el contrario los servlet son aplicaciones java con tags HTML en su interior que deben ser compilados para ser usados.

5.3.1. Pruebas

EjemploJSP.jsp

```
<%@page contentType="text/html"%>
<html>
<head><title>Ejemplo JSP</title></head>
<body>
Ejemplo de Java Server Pages - JSP<br><p>
<%-- Este código no se ejecutara --%>
<% int x=10, y=2;
out.write("El resultado es: " + (x/y));
%>
</body>
</html>
```

(Tabla 5.3.1.1) Ejemplo Jsp

EjemploServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EjemploServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("");
        out.println("<head>Mi primer servlet</head></title>");
        out.println("<i>Ejemplo simple...</i></font>");
        out.println("");
    }
}
```

(Tabla 5.3.1.2) Ejemplo Servlet

5.3.2. Comunicación entre Applet y Servlet

A menudo se requiere este tipo de comunicaciones, las razones son un tanto sencillas. Los Applet se ejecutan en el cliente y los Servlet en el servidor, de esta forma se le encarga al Servlet que realice las labores pesadas de manejo de datos y que solo entregue los datos ya procesados.

Las razones de esta acción es, porque el servidor dispone de mayores recursos que cualquier computador cliente.

El ejemplo siguiente muestra los pasos a seguir, pero no se incluyen en él las medidas de seguridad que aseguren la privacidad de los datos. Una de estas medidas puede ser una simple clave dinámica que solo sea conocida por las dos partes.

ServletPost.java (Applet)

```
package AppletServletPost;
import java.lang.*;
import java.applet.*;
import java.net.*;
import java.io.*;

public class ServletPost extends javax.swing.JApplet {

    public ServletPost() {
        initComponents();
    }

    private void initComponents() {
        // declaración de objetos
        btnConectar = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        txtResult = new javax.swing.JTextArea();
        txtUrl = new javax.swing.JTextField();
    }
}
```

```

btnConectar.setText("Conectar a una página mediante Socket HTTP ");
// Evento Clic
btnConectar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        event_conectar(evt);
    }
});

getContentPane().add(btnConectar, java.awt.BorderLayout.NORTH);

jPanel1.setLayout(new java.awt.BorderLayout());

txtResult.setText("No Conectado...");
jScrollPane1.setViewportView(txtResult);

jPanel1.add(jScrollPane1, java.awt.BorderLayout.CENTER);

txtUrl.setText("http://localhost:8080/tesis/servlet/PrimerServlet");
jPanel1.add(txtUrl, java.awt.BorderLayout.NORTH);
getContentPane().add(jPanel1, java.awt.BorderLayout.CENTER);

}

// Crea un objeto URL con la dirección en la cual se encuentra
// el servlet
private void event_conectar(java.awt.event.MouseEvent evt) {
try {
    txtResult.setText("");
    URL direccion = new URL (txtUrl.getText());
    URLConnection conexion = direccion.openConnection();
    conexion.setUseCaches (false);
    conexion.setRequestProperty ("Accept-Language", "es");
    InputStreamReader stream = new InputStreamReader
(conexion.getInputStream());
    BufferedReader entrada = new BufferedReader(stream);
    String linea;
        // se obtiene todo el buffer de entrada obtenido desde
        // el servlet
    while ((linea = entrada.readLine())!=null){
        txtResult.setText(txtResult.getText() + linea);
    }
    entrada.close();
} catch (Exception ex){
    System.out.println(ex); // es enviado a la Java console
    txtResult.setText(ex.getMessage());
}
}

```

```
}  
  
private javax.swing.JScrollPane jScrollPane1;  
private javax.swing.JTextField txtUrl;  
private javax.swing.JTextArea txtResult;  
private javax.swing.JButton btnConectar;  
private javax.swing.JPanel jPanel1;  
  
}
```

(Tabla 5.3.2) Comunicación Servlet/Applet

Este código puede conectarse a cualquier servlet que se indique en el cuadro de texto y obtener los datos que el retorne.

5.4. Archivos JAR

Un JAR es un formato de archivo empaquetado estándar que permite entre otras cosas ordenar el código precompilado.

Ventajas de un archivo JAR:

- **Compresión:** El formato JAR permite comprimir el contenido para ahorrar espacio.
- **Seguridad:** Es posible firmar digitalmente el contenido de un fichero jar, asegurando de esta forma su procedencia y otorgando privilegio que de otra forma no tendría.

-
- Disminución del tiempo de descarga: Java abre una conexión http para descargar cada archivo. Todos los recursos empaquetados en un JAR son descargados mediante una sola conexión http.
 - Empaquetado: Puede contener todo tipo de archivos como *.class, *.java, *.mid, *.mp3, etc. Usualmente todas las imágenes y sonidos de una aplicación se encuentran dentro de un archivo de este tipo.
 - Portabilidad: El mecanismo para manejar los archivos JAR son parte del estándar de java e independiente de la plataforma operativa.

5.4.1. ¿Cómo crear un archivo JAR?

Crear un archivo jar se asemeja a usar el antiguo compresor ARJ por línea de comando de D.O.S. o el compresor TAR de Linux.

Los parámetros más usados son:

- c: Crea un nuevo contenedor
- v: Genera salida detallada en salida estándar
- f: Especifica nombre de archivo contenedor
- m: Incluir información del archivo manifest especificado

Un Archivo manifest contiene una estructura definida con varios parámetros que indican información sobre el contenido del paquete. El parámetro más usual es Main-Class que indica la clase que se ejecuta por defecto al ejecutar el paquete

completo, de esta forma el paquete funciona como si fuera un archivo ejecutable (*.EXE) multiplataforma.

Manifest-Version: 1.0 Created-By: 1.4.1_02 (Sun Microsystems Inc.) Main-Class: java2d.Java2Demo

(Tabla 5.4.1) Estructura básica del archivo manifest

Ejemplos:

Empaqueta la clase prueba.class

```
# jar cvf nombre_paquete.jar prueba.class
```

Empaqueta el directorio llamado gdat

```
# jar cvf nombre_paquete.jar gdat
```

Empaqueta y agrega el archivo manifest al paquete

```
# jar cvfm GDAT.jar MANIFEST.MF gdat
```

5.4.2. Firmar digitalmente un JAR

Al firmar un paquete se le otorgan posibilidades de acción sobre los recursos del sistema cliente en el cual se ejecuta. Esta opción se utiliza frecuentemente en los Applet, porque estos no tienen acceso a algunos recursos como el disco duro del cliente. Al firmar el Applet se le otorgan todos los privilegios, pudiendo incluso ejecutar comandos como Format.

Para realizar el proceso de firma primero se debe crear una clave de autenticación mediante el comando keytool.

```
keytool -genkey -keystore myKeyStore -alias myself
```

Esta herramienta pide información acerca de la clave y del emisor del certificado que le aparecerá al usuario al ejecutar la aplicación. La clave se almacena en el almacén de claves que se especifique.

La firma del JAR se realiza por medio del comando jarsigner.

```
jarsigner -keystore myKeyStore main.jar myself
```

5.5. Conexión a Base de Datos – JDBC

5.5.1. Introducción

Java utiliza un método de conexión a datos llamado JDBC, el cual es muy similar a ODBC de Windows. Este consiste en un paquete comprimido (JAR) de librerías que hacen de puente entre la aplicación y el motor de datos.

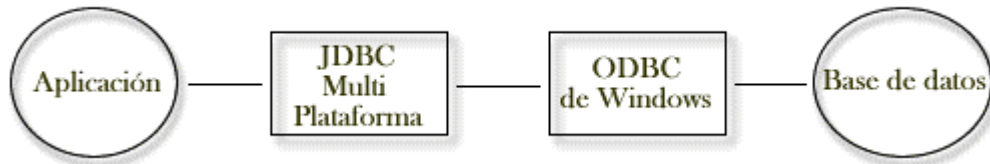
Gracias a esto es posible realizar aplicaciones que se conecten a distintas bases de datos solo cambiando el paquete JDBC por el correspondiente.

5.5.2. Conceptos básicos

Existen dos grandes tipos de conexión que difieren solo en el método de transmisión.



(Figura 5.5.2.1) Conexión JDBC



(Figura 5.5.2.2) Conexión JDBC con ODBC

Los códigos de acceso son estándar indistintamente de la base de datos a utilizar, por lo que puede servir de guía cualquier manual que especifique como realizar la conexión. El sitio de MySQL-hispano tiene disponibles dos excelentes manuales:

- <http://www.mysql-hispano.org/page.php?id=9>

MySQL con Java en Linux

- <http://www.mysql-hispano.org/page.php?id=24>

MySQL con Java en MS Windows

5.5.3. Conexión a Postgresql

El siguientes ejemplo se conecta a una base de datos llamada "mi_bd" y a una tabla llamada "mi_tabla".

```
import java.sql.*;

class PostgreSQLTest {

    public static void main (String args[]) {
        Connection conn = null; //Objeto principal de conexión
        try{
            Class.forName("org.postgresql.Driver"); //Conector JDBC
            //Conexión al servidor de datos
            conn = DriverManager.getConnection("jdbc:postgresql://localhost/mi_bd","postgres","");

            if (conn != null){
                System.out.println("Conexión Ok...");
                Statement stmt = conn.createStatement(); //Crea espacio de trabajo
                stmt.setQueryTimeout(30);
                ResultSet res = stmt.executeQuery("SELECT * FROM mi_tabla ");
                ResultSetMetaData metadata = res.getMetaData(); //Ejecuta la consulta

                while(res.next()){
                    System.out.println(res.getString(1) + "\t" + res.getString(2));
                }
                //Cierra las conexiones
                res.close();
                stmt.close();
                conn.close();
            }
        } catch(SQLException ex){
            System.out.println("SQLException: " + ex.getMessage());
        } catch(ClassNotFoundException ex){
            System.out.println("ClassNotFoundException: " + ex.getMessage());
        }
    }
}
```

(Tabla 5.5.3) Conexión a Postgresql

5.6. Java Web Start - JNLP

Es la capacidad de ejecutar aplicaciones desde un navegador web que sean transparentes para el desarrollador y para el cliente, mediante la cual se realiza el proceso de instalación automática sin mayor intervención por parte del usuario. El control de versiones de las aplicaciones y de dependencias es una de las particularidades que hace que Java Web Start sea una herramienta de gran valor.

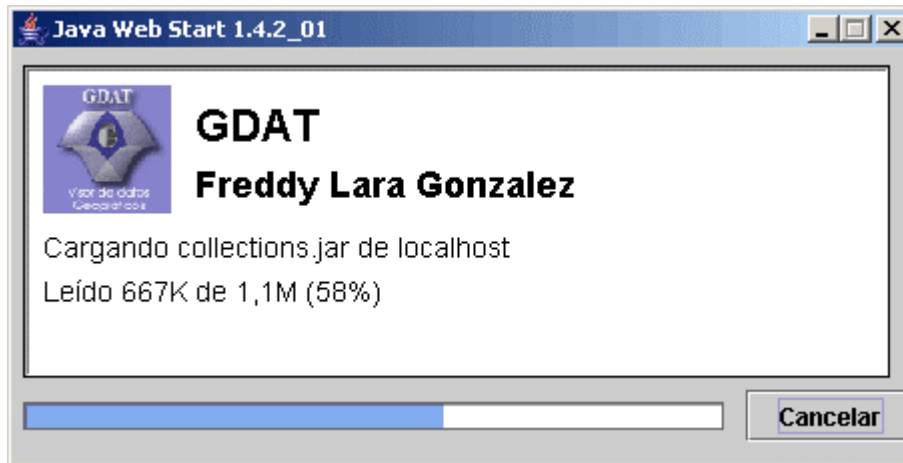
Java Web Start utiliza la especificación JNLP (Java Networking Launching Protocol) que define como ejecutar aplicaciones Java remotamente desde un entorno de red cualquiera.

De esta forma revoluciona el concepto tradicional que se tiene de las aplicaciones. Normalmente cuando se quiere ejecutar una aplicación que no se encuentra instalada en un equipo, se descarga del servidor, se instala en dicho equipo y por último se ejecuta. Java Web Start intenta simplificar al máximo todo este proceso de modo que el usuario lo único que deba hacer para lanzar una aplicación sea simplemente pinchar en un enlace de su navegador, a partir de ese momento, todo el proceso relacionado con la descarga, instalación y ejecución del programa se realiza de una manera transparente.

A pesar de su parecido, una aplicación de Java Web Start no tiene nada que ver con un Applet. Java Web Start sólo utiliza el navegador como medio para que el usuario pueda ejecutar las aplicaciones. Una vez que el usuario pincha

en un enlace de una aplicación, ésta se ejecuta en la máquina virtual del cliente como cualquier otra aplicación.

Estas aplicaciones siguen el modelo de seguridad de la plataforma Java 2 por lo que la integridad de los datos que obtenemos a través de la red está garantizada. Es necesario que las aplicaciones que accedan a los recursos del cliente deban estar debidamente firmadas y se ejecutan solo si el usuario autoriza su ejecución. Java Web Start viene incluido dentro del JRE a partir de su versión 1.4.



(Figura 5.6) Proceso de descarga manejado por Java Web Start

5.6.1. Pruebas

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0+" codebase="http://localhost:8080/jnlp/" href="jnlp.jnlp">

  <information>
    <title>Ejemplo de JNLP</title>
    <vendor>Freddy Lara González</vendor>
    <homepage href="http://www.inf.uct.cl/~flara"/>
    <description> Ejemplo de JNLP </description>
    <description kind="short">
      Configuración de ejemplos sobre el uso de JNLP
    </description>
    <icon href="gdat.jpg"/>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>
    <j2se version="1.4+"/>
    <jar href="jar/aplicacion.jar"/>
    <jar href="jar/libreria1.jar"/>
    <jar href="jar/libreria2.jar"/>
    <extension name="OtroJNLP1" href="otro1.jnlp"/>
    <extension name="OtroJNLP2" href="otro2.jnlp"/>
    <nativelib href="windows/LibreriaDLL.jar"/>
    <nativelib href="linux/LibreriaSO.jar"/>
  </resources>

  <application-desc main-class="org.aplicacion.Main">
  </application-desc>

</jnlp>
```

(Tabla 5.6.1) Ejemplo archivo JNLP

5.6.2.Descripción

La estructura del archivo JNLP es simple e indica todos los archivos necesarios para la ejecución de la aplicación.

La ruta a al archivo debe ser absoluta.

```
<jnlp spec="1.0+" codebase="http://localhost:8080/jnlp/" href="jnlp.jnlp">
```

Los paquetes se agregan de forma descendente.

```
<jar href="jar/aplicacion.jar"/>  
<jar href="jar/libreria1.jar"/>  
<jar href="jar/libreria2.jar"/>
```

En la mayoría de los casos la lista de paquetes es extensa y diversa, por lo que suele agruparse en archivos distintos e incluirlos en el principal.

```
<extension name="OtroJNLP1" href="otro1.jnlp"/>
```

Existen librerías como Java3D que utiliza archivos nativos (*.dll, *.so), se empaquetan en un JAR y se adjuntan.

```
<nativelib href="windows/LibreriaDLL.jar"/>  
<nativelib href="linux/LibreriaSO.jar"/>
```

La clase por defecto se indica con su ruta completa.

```
<application-desc main-class="org.aplicacion.Main">  
</application-desc>
```

Esto asume que existe un archivo Main.class en la ruta org.aplicacion en alguno de los paquetes anteriores.

5.6.3. Tipo MIME

El servidor Web Apache a diferencia del servidor de servlet Tomcat, no reconoce los archivos *.jnlp. Para solucionar esto se agrega al archivo de configuración “/etc/mime.types “ la línea siguiente:

```
application/x-java-jnlp-file jnlp
```

Además sería conveniente también agregar alguno de los siguientes tipos en caso de ser necesario (opcionales):

```
text/vnd.sun.j2me.app-descriptor    jad //Indica un archivo descriptor Micro Edition
application/java-archive            jar //Indica un archivo empaquetado *.jar
text/plain                          java //Indica un archivo de código *.java
application/java                    class //Indica un archivo precompilado *.class
```

5.7. Gráficas

Java provee una gran variedad de funciones gráficas, que se pueden dividir en tres grandes grupos:

- Primitivas: Gráficas simples como líneas, cuadrados, circunferencias, etc.
- En 2 dimensiones (Api Java2D): Ampliación de la anterior sobre la cual se realizan procesos de renderizado sobre imágenes, consiguiendo mejoras considerables.

-
- En 3 dimensiones (Api Java3D): Orientada al modelamiento de mundos virtuales y juegos en tres dimensiones (no incluido en el JDK, se proporciona como librería anexa).

Las funcionalidades proporcionadas por el Api 2D son suficiente para realizar los procesos gráficos implementados durante el transcurso de este este trabajo.

Alguna de estas funcionalidades son:

- Dibujar líneas de cualquier anchura
- Rellenar formas con gradientes y texturas
- Mover, rotar, escalar y recortar texto y gráficos.
- Componer texto y gráficos solapados.
- Modelo de renderizado uniforme para pantallas e impresoras
- Soporte para imprimir documentos complejos.
- Conversión de graficas a archivos.

Por ejemplo, es posible usar el Api 2D para mostrar gráficos y charts complejos que usan varios estilos de línea y de relleno para distinguir conjuntos de datos, como se muestra en la siguiente figura:



(Figura 5.7) Gráficos construidos con Api 2D

5.7.1. ¿Cómo usar Api 2D?

El API 2D presenta `java.awt.Graphics2D`, un nuevo tipo de objeto `Graphics`. `Graphics2D` desciende de la clase `Graphics` para proporcionar acceso a las características avanzadas de renderizado del API 2D de Java.

Para usar las características del API 2D , tenemos que forzar el objeto `Graphics` pasado al método de dibujo de un componente a un objeto `Graphics2D`.

```
public void Paint (Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;  
    ...  
}
```

5.8. Múltiples Hilos y creación de eventos

Java permite la creación de hilos (`Thread`) y múltiples hilos, para lo cual también es necesario conocer el concepto de interface, que permite la creación de eventos.

5.8.1. Múltiples Hilos

Cada aplicación es un hilo, es decir, utiliza un único flujo de control para controlar su ejecución. Sin la necesidad de indicar que se desea utilizar un único flujo, se asume por omisión.

Por ejemplo, en una aplicación estándar de saludo:

```
public class HolaMundo {  
    static public void main( String args[] ) {  
        System.out.println( "Hola Mundo!" );  
    }  
}
```

Cuando se llama a la función main, la aplicación imprime el mensaje y termina. Esto ocurre dentro de un único hilo (Thread).

La creación de múltiples hilos se realiza heredando la clase Thread, cada una de estas instancias se ejecuta de forma paralela al hilo principal en el cual fueron creadas.

Un Thread se inicia al ejecutar el método Start, quien llama de forma interna al método Run en el que definitivamente se encuentra el código que se ejecuta durante la vida del hilo. Éste muere cuando termina de ejecutar el método Run.

Tomando el mismo ejemplo anterior, pero ahora con múltiples hilos:

```
// Clase MiHilo
class MiHilo extends Thread {
    private String nombre;
    private int retardo;

    // Constructor para almacenar el nombre y el retardo
    public MiHilo ( String s, int d ) {
        nombre = s;
        retardo = d;
    }

    // El método run es similar al main, pero para threads.
    // Cuando run termina, el thread muere
    public void run() {
        // Se retrasa la ejecución en el tiempo especificado
        try {
            sleep( retardo );
        } catch( InterruptedException e ) {
            //nada
        }

        // Ahora se imprime el nombre
        System.out.println( "Hola Mundo! "+nombre+" "+retardo );
    }
}

// Clase MultiHola
public class MultiHola {
    public static void main( String args[] ) {
        MiHilo t1,t2,t3;

        // se crean los threads
        t1 = new MiHilo ( "Thread 1",(int)(Math.random()*2000) );
        t2 = new MiHilo ( "Thread 2",(int)(Math.random()*2000) );
        t3 = new MiHilo ( "Thread 3",(int)(Math.random()*2000) );

        // Se inicia la ejecución de los threads
        t1.start();
        t2.start();
        t3.start();
    }
}
```

(Tabla 5.8.1) Ejemplo de multihilo

Al ejecutar este código, imprime los mensajes pero en desorden. Al pasarle el parámetro retardo de forma totalmente aleatoria no se sabe con certeza el orden de impresión de los mensajes.

- Ventajas de usar múltiples hilos

Una aplicación que realice procesos pesados (de entrada y salida, modelamiento matemático, renderización de imágenes, etc) consumirá una gran cantidad de recursos del sistema, incluso es posible que el equipo no responda por algunos segundos. Al encapsular estos procesos dentro de un Thread se ejecuta de forma paralela (en segundo plano) intercambiando el uso del procesador de forma mucho más fluida con el resto de las aplicaciones.

5.8.2. Creación de eventos

Si un proceso con múltiples hilos descarga archivos desde un servidor remoto, solo es posible saber el momento en el cual comienza el proceso de descarga, pero no así el momento en el que finaliza, dado que se ejecuta de forma independiente en su propio flujo de control.

El uso de interfaces es primordial para crear eventos dentro de un hilo que permita informar el estado del mismo. Una interfaz define sólo el encabezado de los métodos, pero no define las acciones o código que contiene cada uno de ellos.

El código de éstos métodos se define en la clase que implementa la interfaz, de esta forma:

```
// La interfaz define 2 métodos
public interface MiInterfaz {
    public void AccionIniciada();
    public void AccionTerminada();
}

//La clase implementa el código de los métodos
class MiClase implements MiInterfaz {
    //Implementa el método
    public void AccionIniciada(){
        // Código
    }

    //Implementa el método
    public void AccionTerminada(){
        // Código
    }
}
```

(Tabla 5.8.2.1) Ejemplo de creación de interfaces

Para obtener los eventos particulares desde otras clases, en este caso desde un Thread, se crea un puntero que indica donde se encuentra el código que se ejecuta, es decir, una variable que contiene la referencia a la clase que implementa la interfaz.

Por ejemplo:

```
private MiInterfaz handler;
```

Tomando los ejemplos anteriores, se definen algunos eventos particulares:

```
// La interfaz define 2 métodos
public interface MiInterfaz {
    public void AccionIniciada();
    public void AccionTerminada();
}

// Clase MiHilo
class MiHilo extends Thread {
    private String nombre;
    private int retardo;
    // Define una referencia a un objeto vacío del tipo MiInterfaz
    private MiInterfaz handler=null;

    public MiHilo (MiInterfaz interfaz, String s, int d ) {
        // Define la referencia a un objeto que implemente MiInterfaz
        handler= interfaz ; nombre = s; retardo = d;
    }

    public void run() {
        // llama al método AccionIniciada contenido en el objeto
        // indicado por la variable handler
        handler.AccionIniciada();

        try {
            sleep( retardo );
        } catch( InterruptedException e ) { }
        System.out.println( "Hola Mundo! "+nombre+" "+retardo );

        // llama al método AccionTerminada contenido en el objeto
        // indicado por la variable handler
        handler.AccionTerminada ();
    }
}

// Clase MultiHola que implementa la interfaz
public class MultiHola implements MiInterfaz {
    public static void main( String args[] ) {
        private MiHilo t1;

        // se crea el thread, la variable this apunta el objeto actual, es decir, MultiHola
        t1 = new MiHilo ( this, "Thread 1", (int)(Math.random()*2000) );
    }
}
```

```
// Se inicia la ejecución del thread
t1.start();
}

// Evento que indica el momento en el se inicia la ejecución del hilo
public void AccionIniciada(){
    System.out.println( "Se inició la ejecución del hilo" );
}

// Evento que indica el momento en el se termina la ejecución del hilo
public void AccionTerminada(){
    System.out.println( "Se termino la ejecución del hilo" );
}
}
```

(Tabla 5.8.2) Ejemplo de creación de eventos dentro de un multihilo

6. GEOTOOLS

6.1. Introducción

Geotools es una librería orientada al desarrollo de aplicaciones SIG sobre plataformas Java. Provee métodos que permiten manipular la información geográfica obtenida y además dibujar las geometrías vectoriales.

Actualmente están disponibles dos versiones llamadas Geotools 1 (GT1) y Geotools 2 (GT2) las cuales presentan características distintas.

La primera (GT1) tiene como objetivo manipular archivos shape, al igual que la segunda (GT2) por razones de compatibilidad, además ésta última está orientada a la conectividad de datos bajo OpenGis.

Ambas librerías están actualmente en continuo desarrollo, los fuentes, binarios, documentación y demos se encuentran disponibles en www.geotools.org.

6.2. Geotools 1 (GT1)

Está compuesta por tres paquetes:

- geotools.jar
- collections.jar
- support.jar

Además es posible encontrar algunos ejemplos sencillos sobre su uso y algunos mapas para comenzar a programar:

<http://multilingual.geotools.org/resources/>

Geotools 1 esta orientado al uso de archivos shapefiles (*.shp) que se usa en Arcview. Por lo que también incluye soporte para los archivos *.dbf que contienen información referente a las geometrías.

Las librerías que forman parte de este paquete son aproximadamente un 80% más livianas que las usadas en Geotools 2. En las pruebas realizadas, y disponibles en www.inf.uct.cl/~flara/prototipos.htm se ve claramente que la demo desarrollada con GT1 pesa 775 KB y la demo realizada con GT2 pesa 7,92 MB (incluidas las librerías gráficas JavaJAI). Lo que hace pensar que GT1 es más portable sobre internet en aplicaciones como Applet por el menor tiempo de descarga que esto implica.

6.3. Geotools 2 (GT2)

Es un gran conjunto de librerías, de las cuales solo están disponibles los fuentes. El proceso de compilación es algo simple, pero puede transformarse en algo extremadamente complejo si no se tiene una idea clara de los pasos a seguir.

Una vez compilados se obtienen 30 archivos JAR de los cuales 11 son necesarios para generar cualquier tipo de aplicación:

- core-SNAPSHOT.jar
- cts-coordtrans-SNAPSHOT.jar
- data-SNAPSHOT.jar
- defaultcore-SNAPSHOT.jar
- gui-SNAPSHOT.jar
- lite-rendering-SNAPSHOT.jar
- opengis-SNAPSHOT.jar
- resources-SNAPSHOT.jar
- shapefile-SNAPSHOT.jar
- sldstyling-SNAPSHOT.jar
- utils-SNAPSHOT.jar

Además, se requiere de tres paquetes adicionales para que los paquetes anteriores funcionen, estos son parte del repository de maven y se obtienen durante el proceso de compilación.

- JTS-1.2.jar
- JTS-1.3.jar
- vecmath-1.2.1.jar

GT2 presenta una característica más que la hace diferente de la GT1, y es que utiliza para dibujar las geometrías, librerías de manejo de imagen de java llamadas JavaJAI que se pueden obtener en el sitio de SunMicrosystem (<http://java.sun.com/products/java-media/jai/current.html>).

El proceso detallado de la compilación de los paquetes se especifica en el anexo.

6.4. ¿Cómo utilizar Geotools?

A modo de introducción a las librerías Geotools, se presenta una pequeña reseña de las clases, métodos y funcionalidades básicas y necesarias para implementar un desarrollo SIG. Todos los comentarios siguientes están basados en las pruebas realizadas en el anexo e.

6.4.1. Tutorial de Geotools 1

EL código completo de la prueba que se detalla a continuación se encuentra en el anexo e.1.

Primero hay que indicar los paquetes de Geotools que se van a utilizar.

```
import uk.ac.leeds.ccg.geotools.*;  
import uk.ac.leeds.ccg.widgets.*;  
import uk.ac.leeds.ccg.shapefile.*;
```

Estos proporcionan los objetos de manejo de archivos shapes y herramientas graficas por medio de las cuales los mapas toman forma.

El principal objeto es Viewer, que es el componente encargado de esquematizar las representaciones geométricas, es decir, dibujar los mapas.

ShapefileReader se encarga de ejecutar la carga de datos desde los *.shp y *.dbf.

También acepta *.zip, el que debe contener en su interior un shp y dbf con el mismo nombre del zip. La carga se realiza a través de una dirección URL, todas las direcciones locales pueden ser transformadas a direcciones URL de la misma forma.

```
private Viewer view;  
private ShapefileReader loader;
```

Se crea una nueva instancia del objeto y se agrega en este caso a un objeto JPanel llamado PanelDibujo.

```
view = new Viewer();  
PanelDibujo.add(view, "Center");
```

Esta nueva instancia no contiene dato alguno. La carga de datos se realiza por medio de ShapefileReader, al que se le proporciona la URL, cabe notar que el nombre de archivo contenido en File no contiene extensión, por lo que asume extensión zip.

```
String File = "maps/buildings";  
String base= "http://www.inf.uct.cl/~flara/";  
  
try{  
  
    loader = new ShapefileReader(new URL(base,File));  
}  
catch(MalformedURLException e){  
    System.out.println("Error al cargar el archivo shapefile "+ File +": "+e);  
}
```

```
}
```

La apariencia y atributos de un mapa se obtienen a través de un archivo *.key que no pertenece a la especificación SIG, mas bien, es una adecuación por conveniencia que se encarga de cargar los colores correspondientes para cada uno de los objetos geométricos contenidos en el Shapefile.

```
Shader shader;  
String Lut="maps/buildings.key";  
try{  
    shader = new DiscreteShader(new URL(base,Lut));  
}  
catch(IOException e){  
  
    System.err.println("Error al cargar el archivo key "+ Lut );  
    shader = new MonoShader(Color.gray);  
}
```

En caso error de carga, se crea un una apariencia gráfica monocromática, en este caso de color gris.

Un archivo key contiene una estructura simple. Es una serie de registros compuesta de tres valores: Campo Geométrico (Index), Color (estilo web), Texto.

1,'#f0f0c0',Shop
2,'#101080',Commercial
3,'#6666cc',Food/Drink
4,'#c0c0f0',Industrial
5,'#c0f0f0',Hotel

(Tabla 6.4.1) Estructura del archivo Key

Las luces de selección se activan cuando el ratón se encuentra sobre un área del mapa (objeto cerrado).

```
Theme theme;  
HighlightManager highlightManager= new HighlightManager();  
theme.setHighlightManager(highlightManager);
```

Luego de esto se crea el objeto Theme que contiene las características de apariencia más los datos geométricos.

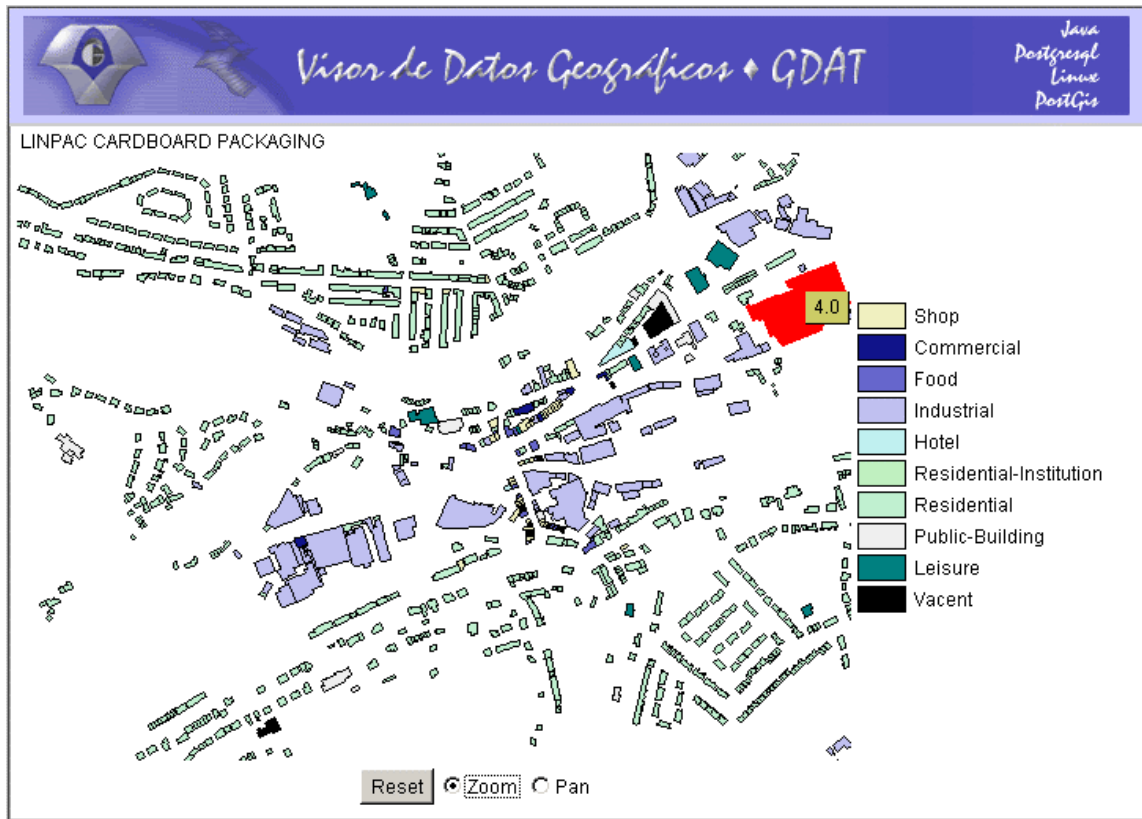
```
String ShaderBy="use";  
theme = loader.getTheme(shader,ShadeBy);  
view.addTheme(theme);
```

Donde ShaderBy corresponde a un conjunto de shapes designados por el nombre de "use". Esta variable, en este caso, solo es un nombre de una columna de datos que a su vez tiene cierta cantidad de registros asociados.

Luego el theme se agrega al viewer, quien toma la información de éste para realizar el dibujo. Un mapa puede estar compuesto por varias coberturas, que corresponden cada una a un Shapefile distinto, por lo que cada cobertura, se agrega de forma individual al viewer de la misma forma.

Las herramientas de paneo y zoom se crean dando como referencia el objeto viewer creado anteriormente y que luego se agrega al JPanel.

```
ToolBar controls = new ToolBar(view);  
PanelDibujo.add(controls,"South");
```



(Figura 6.4.1) Demo Geotools 1

6.4.2. Tutorial de Geotools 2

EL código completo de la prueba que se detalla a continuación se encuentra en el anexo e.2.

Primero hay que indicar los paquetes de Geotools que se van a utilizar.

```
import org.geotools.data.shapefile.ShapefileDataSource;  
import org.geotools.data.*;  
import org.geotools.gui.swing.MapPaneImpl;  
import org.geotools.gui.swing.ToolMenu;  
import org.geotools.styling.Style;  
import org.geotools.styling.StyleBuilder;  
import org.geotools.map.*;  
import org.geotools.util.*;  
import org.geotools.ct.*;  
import org.geotools.feature.*;  
import org.geotools.factory.*;
```

El componente principal es MapPanelImpl el cual se encarga de realizar los dibujos.

Al igual que GT1, primero hay que realizar la carga de los archivos a través de la URL. ShapefileDataSource se encarga de este proceso.

```
private static FeatureCollection fc;  
String AppPathResource="http://www.inf.uct.cl/~flara/";  
ShapefileDataSource sds;  
sds = new ShapefileDataSource(new URL(AppPathResource +"maps/statepop.shp"));  
fc = sds.getFeatures();
```

Luego se crea el manejador de estilos Style.

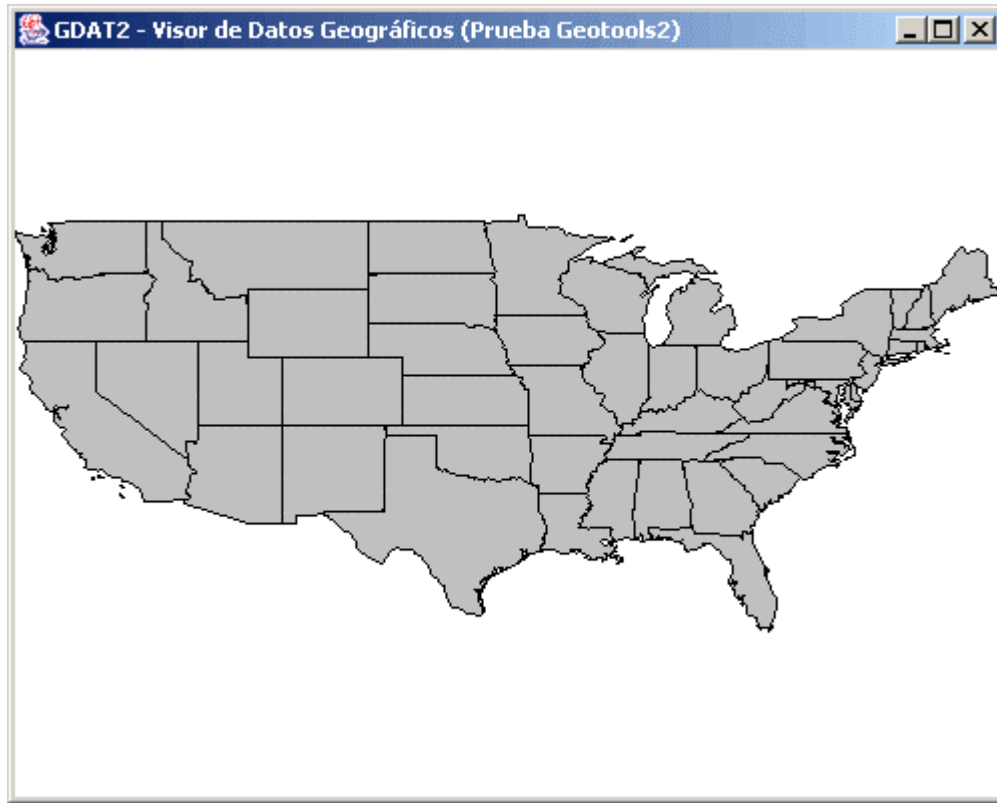
```
private static Style simple;  
StyleBuilder sb = new StyleBuilder();  
simple = sb.createStyle(sb.createPolygonSymbolizer(Color.LIGHT_GRAY,  
Color.BLACK, 1));
```

Se cargan las coberturas obtenidas desde el Shapefile junto con el manejador de estilo sobre ContextFactory.

```
Layer layer;  
ContextFactory contextFactory = ContextFactory.createFactory();  
context = contextFactory.createContext();  
layer = contextFactory.createLayer(fc, style);  
layer.setTitle("Test layer");  
context.getLayerList().addLayer(layer);
```

Para luego cargarlas a MapPanelImpl quien finalmente realiza la tarea del dibujo.

```
MapPanelImpl mapPane;  
mapPane = new MapPanelImpl(context);  
mapPane.setBackground(Color.WHITE);  
  
mapPane.setPreferredSize(new Dimension(300, 300));
```



(Figura 6.4.2) Demo Geotools 2

7. VISOR DE DATOS GEOGRÁFICOS

7.1. Introducción

El desarrollo de este tema, en un principio, toma distintos caminos, así como también abarca distintas tecnologías (como se indica en los capítulos anteriores).

En general, el tema es en sí una investigación, puesto que cada una de las posibles soluciones al problema puntual, representan conocimientos que se adquirieron durante el proceso mismo.

Para cerrar el capítulo, se presenta la implementación del Visor de datos geográfico, explicando los aspectos más relevantes del mismo.

7.2. Etapas de la investigación

Inicialmente se adquieren conocimientos sobre el lenguaje Java (capítulo 5) y se realizan algunas pruebas con el fin de ver las bondades y limitaciones del lenguaje y el resto de las herramientas.

Con esto se tiene la base sobre el cual tomar las primeras decisiones hacia una solución:

Para lograr una solución en línea lo primero es pensar en desarrollar un Applet (capítulo 5.2), puesto que se ejecuta en el navegador de Internet y tiene total capacidad para descargar archivos, en este caso mapas.

Para el almacenamiento de los mapas existen 2 tipos (existen otros formatos pero no son mencionados en este trabajo), en archivos Shapefiles y en una estructura de base de datos OpenGis. Y se realizan pruebas sobre ambas estructuras (capítulo 4).

El servidor de datos usado es Postgresql con PostGis (Capítulo 4.5.1, anexo a, anexo b) sobre Linux Red Hat 9, donde se comienza por detallar algunas funcionalidades del motor y sobre el uso de PostGis.

Para levantar los datos desde el Applet o desde cualquier aplicación java, se utiliza JDBC (capítulo 5.5). JDBC es un paquete que permite la conectividad a bases de datos, este paquete tiene un tamaño variable dependiendo del Driver que contenga y de la versión.

No conforme con el rendimiento obtenido en las pruebas, dado que el Applet solo puede conectarse al servidor de origen, o sea del cual es descargado, y dado que el conector JDBC (aproximadamente 300kb) se descarga junto con el Applet cada vez, hace que la aplicación tarde algo en descargarse.

Teniendo en cuenta esto, y con el afán de optimizar el tiempo de descarga y agilizar todos los procesos es que se intenta realizar una comunicación entre Applet y Servlet (capítulo 5.3), de esta forma se omite la descarga del conector JDBC, porque el procesamiento de datos se realiza en el servidor y no el cliente.

Por esta razón es que se utiliza el servidor de servlet Tomcat (anexo c), y se fusiona con el servidor web Apache para funcionar por el puerto 80.

Luego de experimentar con gráficas simples y sus extensiones llamadas Java2D (capítulo 5.7), se observa que el manejo gráfico necesario para dibujar los datos espaciales a escala y que pueden ser de distintos tipos de geometrías sería una tarea más que extensa.

Por esta razón se prueba el proyecto Geotools (capítulo 6, anexo d) que es un conjunto de librerías SIG, el que dispone de dos versiones Geotools1(GT1) y Geotools2(GT2). La primera trabaja sobre Shapefiles y la segunda sobre OpenGis. Las pruebas de GT2 no fueron lo esperado, porque esta librería está aún en desarrollo y la documentación no es buena.

Esta es la razón principal por la cual se decide desechar GT2 y continuar con GT1 para realizar el prototipo final.

Los Applet descargan todos los paquetes (*.jar, *.class) que necesitan para su funcionamiento cada vez que se ejecutan. Lo que hace que el proceso de ejecución se torne un tanto lento cuando los paquetes son de un tamaño considerable.

En el caso particular, GT1 está formado por 3 paquetes que en conjunto suman 593kb más la aplicación y otros paquetes adicionales, puede fácilmente llegar a pesar 1 MB, cuyo tiempo de descarga es considerable. Además los Applet no poseen un indicador que informe en tiempo real el estado de descarga actual.

Debido a esto se opta por reemplazar los Applet por una aplicación con ventanas (JFrame), cuyo proceso de descarga es controlado por medio de Java Web Start (capítulo 5.6). Gracias a esto el proceso de descarga se realiza una única vez, además las actualizaciones son descargadas de forma automática de la misma forma.

Esta transformación no altera en nada el objetivo del prototipo (Desplegar mapas en línea). Todo lo contrario optimiza el proceso de carga, dado que el Applet descarga la totalidad de los paquetes usados cada vez y Java Web Start solo lo realiza una vez, mejorando el tiempo de respuesta y el despliegue de los mapas sigue siendo en línea, Java es un lenguaje orientado a Internet, de esta forma siempre se obtiene una solución óptima.

7.3. Análisis y especificación de requerimientos

A continuación y en forma breve, se explican los aspectos que enmarcan el análisis y especificación de requerimientos del “Visualizador de datos geográficos a través de Internet”.

7.3.1. Requisitos Funcionales

El Visor esta orientado a proveer de una interfaz de usuario por medio de la cual permita visualizar gráficas vectoriales desde cualquier equipo conectado a la red de Internet y obtener información relevante sobre los mapas y tomar decisiones a partir de los mismos. Los mapas al igual que el Visor, serán descargados desde un servidor remoto dispuesto previamente para aquello.

Además, debe proveer de las herramientas que permitan realizar acciones comunes sobre los datos como lo son:

- Escalado de gráficas: Aumentar o disminuir el tamaño de la imagen.
- Imprimir: Usar la impresora local para imprimir las gráficas.
- Exportar: Convertir a archivo de imagen.
- Nivel de detalle: Indicar información sobre cada cobertura y sus datos adjuntos.

7.3.2. Requisitos no Funcionales

Los requisitos no funcionales se describen a continuación, estos son aspectos que no forman parte de la aplicación pero que se encuentran presentes.

7.3.2.1. Documentación

La documentación será orientada principalmente a los desarrolladores futuros, es decir, entregará detalles técnicos comprensibles solo para otros desarrolladores. Detallará los métodos y clases usadas en el desarrollo, de forma de entregar un mayor nivel de detalle para usuarios interesados en ampliar o mejorar las opciones del Visor.

7.3.2.2. Consideraciones de Hardware

Las aplicaciones Java tienen la particularidad de correr en cualquier computador con Java Virtual Machine (JVM) instalado. Por lo que no se requiere un hardware específico, solo es necesario un computador con conexión a Internet.

7.3.2.3. Ambiente Físico

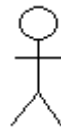
La Universidad Católica de Temuco dispondrá de un sitio web sobre el cual se tendrá acceso a la aplicación y los recursos adjuntos.

7.3.3. Modelos del Sistema

De forma de esquematizar el análisis funcional, se presenta el Modelo de Casos de Uso. Este modelo muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

7.3.3.1. Componentes del Modelo de Casos de Uso

- Actores: Un actor es cualquier entidad, como una persona u otro sistema y que realiza algún tipo de interacción con el sistema.



(Figura 7.3.3.1.1) Representación de un actor.

- Casos de uso: Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

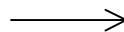


(Figura 7.3.3.1.2) Representación de un caso de uso.

-
- Relaciones: Indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple. Las relaciones pueden ser de distintos tipos, entre ellas encontramos las siguientes :

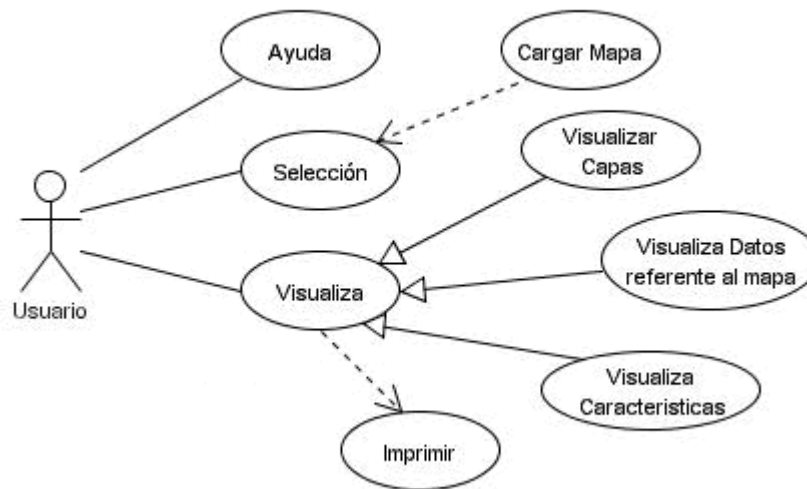
<incluye> : Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso.

<extiende> : Cuando un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita).



(Figura 7.3.3.1.3) Representación gráfica de una relación.

7.3.3.2. Modelo de casos de uso



(Figura 7.3.3.2) Modelo de casos de uso

7.3.3.3. Descripción de casos de uso

A continuación se presenta la descripción extendida de los casos de uso, en ella encontramos la descripción detallada de cada caso de uso utilizado en el modelo de caso de uso presentados anteriormente, su funcionamiento y los detalles asociados que serán requerimientos para desarrollar la aplicación.

CASO DE USO :	Selección
Descripción	Provee de una lista de los mapas disponibles en el servidor remoto para posteriormente ser seleccionados.
Actores	Usuario.
Precondición 1	El cliente debe establecer una conexión mediante Internet con el sistema para activar este caso de uso. Al no existir ésta conexión no estarán disponibles los mapas.
Flujo normal	1.- El caso de uso se activa al momento de iniciar la aplicación y se establece la conexión. 2.- Se obtiene la ruta a cada una de las coberturas dispuestas para cada mapa. 3.- Se muestra el nombre representativo del mapa.

Flujo alternativo 1	En el paso 1, si la conexión no se establece, el sistema envía un mensaje de error al intentar de acceder al recurso remoto.
Post Condiciones	La conexión entre el cliente y el sistema debe permanecer abierta durante el uso del sistema, específicamente al momento de seleccionar un nuevo mapa.

(Tabla 7.3.3.3.1) Caso de uso: Selección.

CASO DE USO :	Cargar Mapa
Descripción	Descarga desde el servidor remoto las coberturas correspondientes al mapa seleccionado y las almacena en memoria para ser utilizadas sólo mientras no se seleccione un nuevo mapa o se cierre el sistema.
Actores	Usuario.
Precondición 1	El caso de uso “Selección” debe listar al menos un mapa.
Flujo normal	1.- El caso de uso es activado por el usuario una vez que es elegido un mapa. 2.- Se comienza el proceso de descarga de las coberturas. 3.- Una vez terminado el proceso se oculta la ventana de progreso de descargas.
Flujo alternativo 1	El proceso de descarga es cancelado por el usuario.
Flujo alternativo 2	El proceso de descarga es cancelado por el sistema debido a un error, en cuyo caso envía el mensaje correspondiente.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.2) Caso de uso: Cargar Mapa.

CASO DE USO :	Visualiza
Descripción	Provee de la interfaz sobre la cual se cargan las distintas propiedades del mapa y dibuja las geometrías.
Actores	Usuario.
Precondición 1	El caso de uso “Cargar Mapa” debe ejecutarse de forma correcta.
Flujo normal	1.- El caso de uso es activado en el momento que se obtienen las coberturas desde el caso de uso “Cargar Mapa”. 2.- Se dibujan las geometrías correspondientes para cada cobertura.
Flujo alternativo 1	No existe flujo alternativo.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.3) Caso de uso: Visualiza.

CASO DE USO :	Visualiza Capas
Descripción	Lista las coberturas existentes para el mapa seleccionado.
Actores	Usuario.
Precondición 1	El caso de uso “Visualiza” debe ejecutarse de forma correcta.
Flujo normal	1.- Se inicia una vez dibujadas las geometrías. 2.- Muestra el nombre de la cobertura permitiendo la visibilidad o invisibilidad de la misma según selección.
Flujo alternativo 1	No existe flujo alternativo.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.4) Caso de uso: Visualiza Capas.

CASO DE USO :	Visualiza Datos Referente al Mapa
Descripción	Permite obtener los datos contenidos en cada uno de los archivo DBF para ser analizados.
Actores	Usuario.
Precondición 1	El caso de uso “Visualiza” debe ejecutarse de forma correcta.
Flujo normal	1.- El usuario abre la ventana de selección de coberturas. 2.- Se cargan los registros asociados a la cobertura.
Flujo alternativo 1	No existe flujo alternativo.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.5) Caso de uso: Visualiza datos referente al mapa.

CASO DE USO :	Visualiza Características
Descripción	Permite obtener información detallada sobre la leyenda de cada cobertura.
Actores	Usuario.
Precondición 1	El caso de uso “Visualiza” debe ejecutarse de forma correcta.
Flujo normal	1.- Se inicia una vez dibujadas las geometrías. 2.- Muestra el detalle de colores correspondientes a la leyenda.
Flujo alternativo 1	No existe flujo alternativo.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.6) Caso de uso: Visualiza características.

CASO DE USO :	Imprimir
Descripción	Permite imprimir las gráficas actuales por la impresora local.
Actores	Usuario.
Precondición 1	El caso de uso “Visualiza” debe ejecutarse de forma correcta, además necesita una impresora local.
Flujo normal	1.- El usuario lanza el caso de uso por medio de la interfaz principal. 2.- Seleccionar la impresora local. 3.- Imprime las gráficas actuales.
Flujo alternativo 1	El caso de uso es cancelado por el usuario.
Post Condiciones	No existen post condiciones para este caso de uso.

(Tabla 7.3.3.3.7) Caso de uso: Imprimir.

CASO DE USO :	Ayuda
Descripción	Proporciona información relevante para el futuro desarrollador. Guiar obligada para facilitar la ampliación de la aplicación.
Actores	Usuario.
Precondición 1	No existe precondición.
Flujo normal	1.- El usuario inicia el caso de uso desde la interfaz principal. 2.- Muestra el detalle de las clases que forman parte del código fuente de la aplicación.
Flujo alternativo 1	No existe flujo alternativo.
Post Condiciones	No existen post condiciones para este caso de uso.

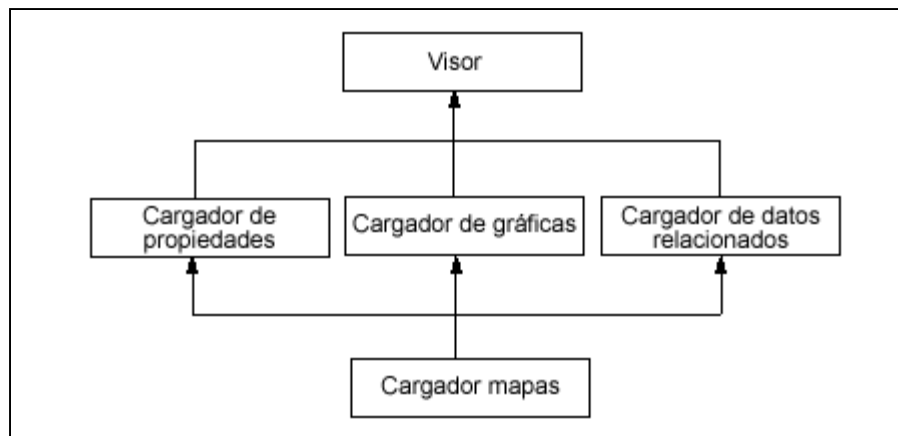
(Tabla 7.3.3.3.8) Caso de uso: Ayuda.

7.4. Análisis

Se enfoca en la producción del modelo de análisis, es decir, en la estructuración y formalización de los requerimientos obtenidos de los usuario. Ayuda a los desarrolladores a verificar la especificación producida durante la obtención de requerimientos.

7.4.1. Modelo de generalización









Permite organizar conceptos en jerarquías. En la parte superior se encuentra la parte general y en la parte inferior de la jerarquía están los conceptos mas especializados.



(Figura 7.4.1) Modelo de generalización



7.4.2. Descripción general del modelo de clases de análisis







7.4.2.1. Interfaces

 Lista de mapas	Formulario que lista los mapas disponibles	 Lista de ayuda	Formulario que muestra la ayuda
 Proceso de carga	Formulario que indica el proceso de carga actual de los mapas	 Lista de capas	Formulario muestra las capas o coberturas disponibles
 Registros	Formulario de lista los datos asociados	 Características	Formulario muestra las características gráficas
 Impresora	Impresora que se encuentra en el cliente	 Dibujo vectorial	Formulario que representa de forma gráfica las geometrías

(Tabla 7.4.2.1) Interfaces

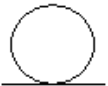


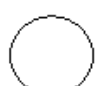
7.4.2.2. Procesos

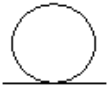
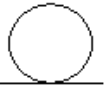
 Carga lista de Mapa	Carga la lista de mapas disponibles desde el servidor remoto	 Ayuda	Carga la ayuda disponible
--	--	---	---------------------------

 Descargar Mapas	Descarga los archivos shapes desde el servidor remoto	 Dibujar	Interpreta y dibuja los datos referenciados
 Cargar registros	Carga todos los registros asociados a los datos espaciales	 Imprimir	Imprime el mapa actual
 Cargar Características	Carga características de estilo gráfico que identifican al mapa	 Carga capas	Carga las capas disponibles para el mapa actual

(Tabla 7.4.2.2) Procesos

7.4.2.3. Entidades

 Biblioteca de mapas	Conjunto de archivos *.shp, *.dbf y *.key que se encuentran en el servidor remoto	 Ayuda	Lista de ayuda disponible para el desarrollador
 Directorio	Archivo maps.dat que contiene las rutas absolutas a los mapas y sus coberturas	 Layers	Conjunto de archivos *.shp que contienen las geometrías.

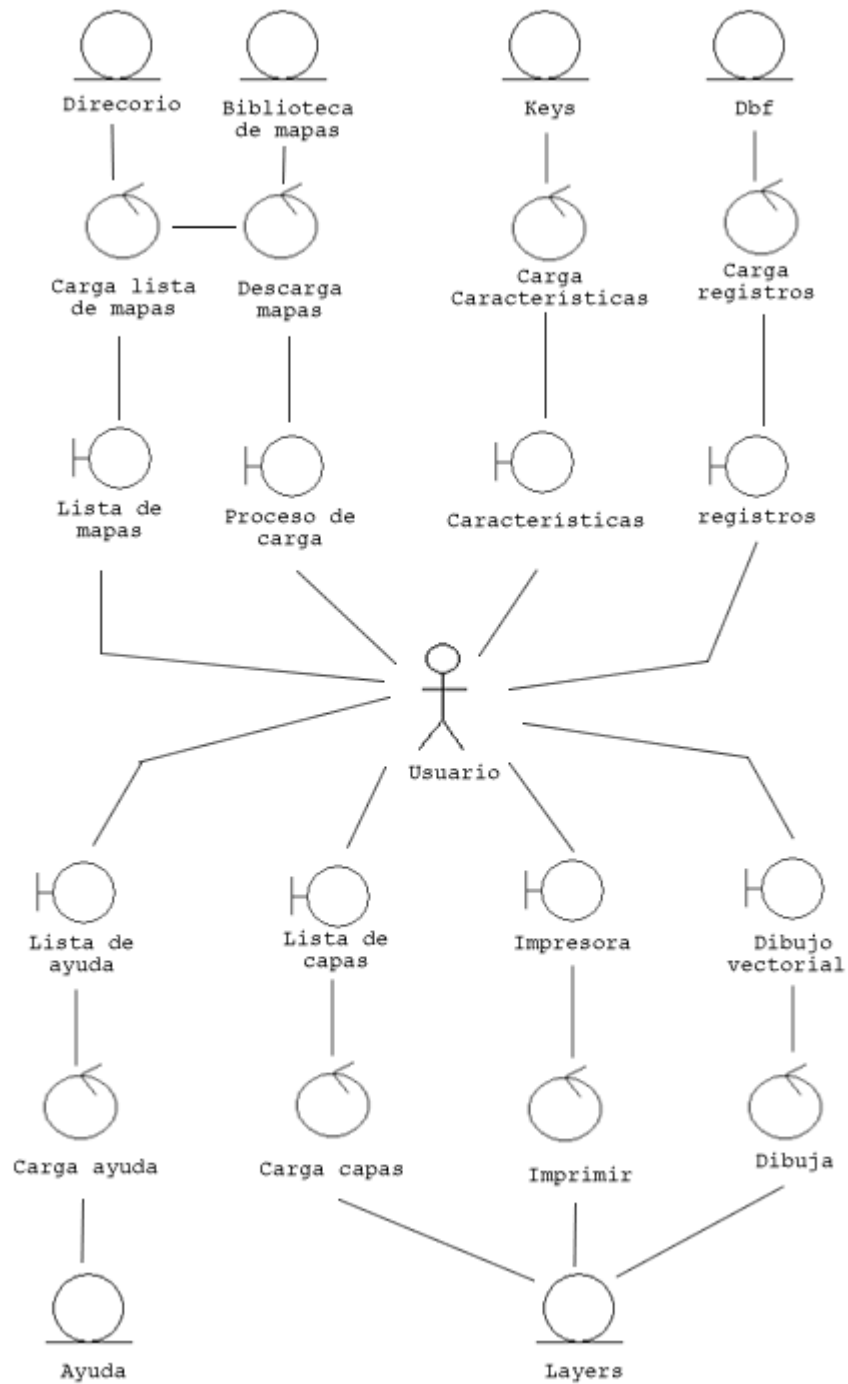
 <p>Dbf</p>	 <p>Keys</p>
--	--

Conjunto de archivos de *.dbf que contienen los datos asociados a las geometrías

Conjunto de archivos de *.key que contienen las características gráficas asociadas a los Layers

(Tabla 7.4.2.3) Entidades

7.4.3. Modelo de análisis de la aplicación



(Figura 7.4.3) Modelo de análisis de la aplicación

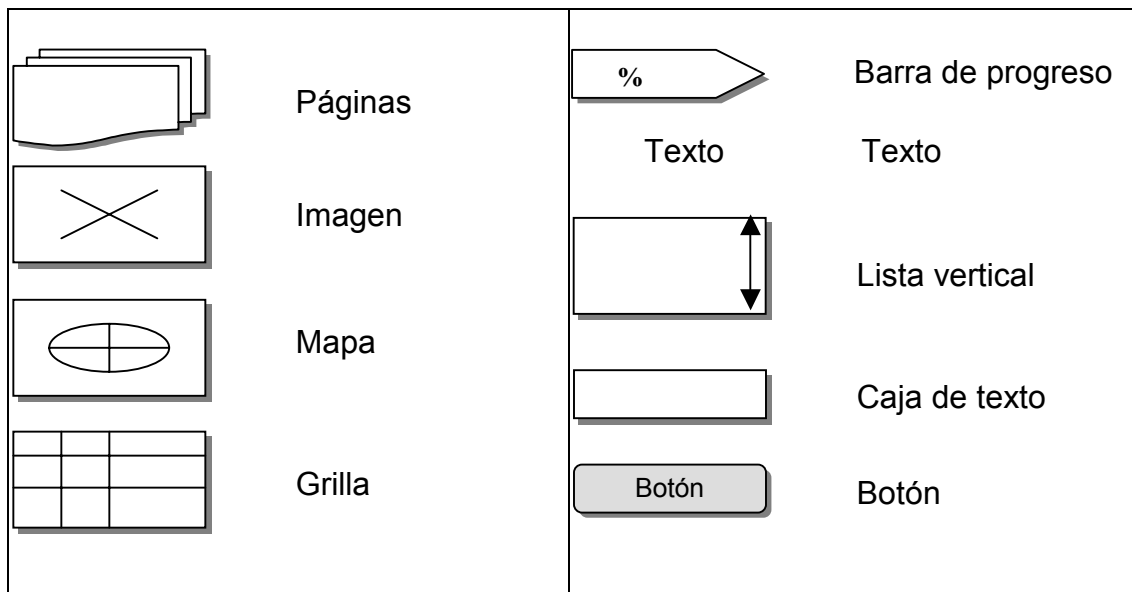
7.5. Diseño

Con la información obtenida en el modelo de análisis se definen los objetivos de diseño del sistema y se descompone en subsistemas para lograr su implementación.

7.5.1. Diseño de interfaces

La interfaz de usuario es representada por medio elementos básicos necesarios para implementar el Visor.

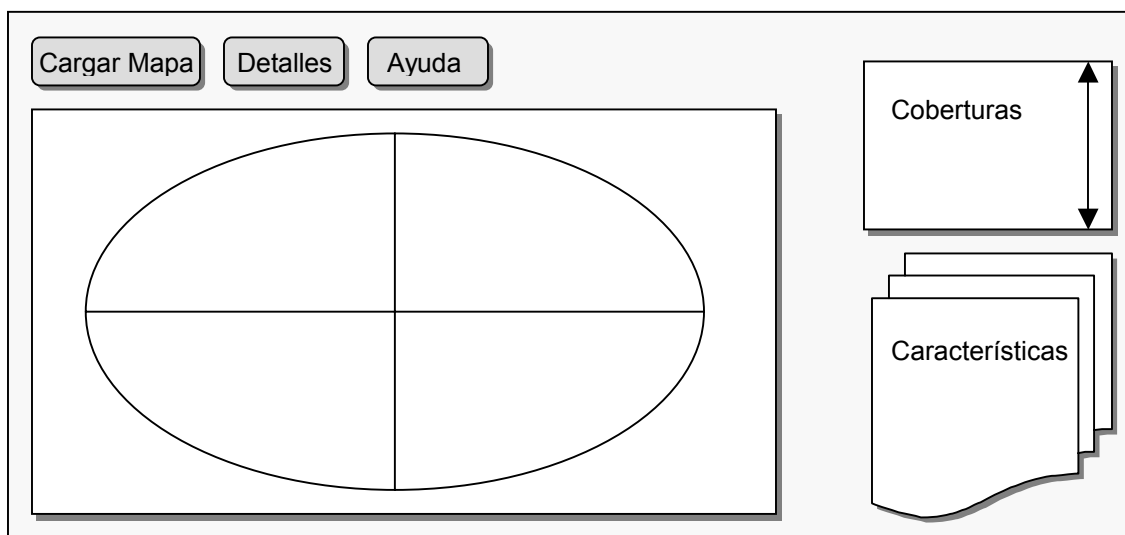
Elementos de interfaces



(Figura 7.5.1.1) Elementos de interfaces

-
- Pantalla principal

En la cual se muestra el mapa en el centro del formulario y en su costado la lista de coberturas y características que posee el mapa.



(Figura 7.5.1.2) Pantalla inicial

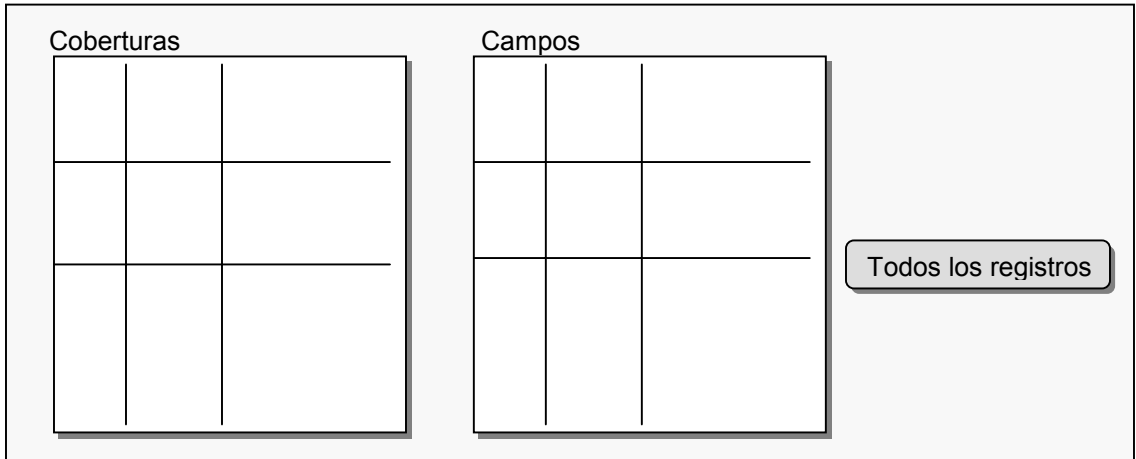
Esta pantalla es el centro de la aplicación y será la encargada de iniciar la carga de datos a los distintos componentes desde la Biblioteca de Mapas. En el costado superior derecho se muestra la lista de coberturas (capas) obtenidas del conjunto archivos de geometrías (Layers). El usuario podrá seleccionar un subconjunto de estas para apreciar por separado su contenido.

La características, en este caso el detalle de colores más su descripción se obtienen a partir del conjunto de archivos Key.

En el centro de la pantalla se muestra la gráfica correspondiente a la representación vectorial de las geometrías, es decir, el mapa.

-
- Pantalla de detalles

Lista la totalidad de los registros asociados a las geometrías.



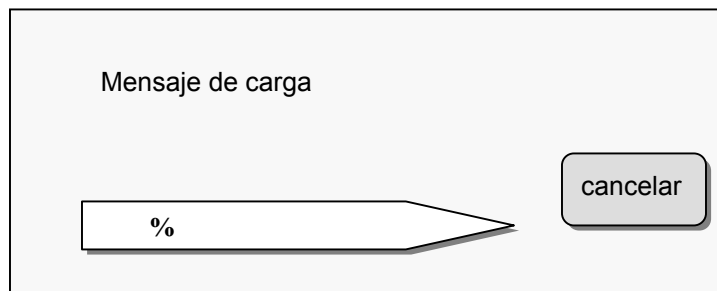
(Figura 7.5.1.3) Pantalla detalles

El conjunto de archivos Dbf contiene la información referente a las geometrías de donde se extraerán los datos. La pantalla estará compuesta de dos grillas.

La grilla Coberturas le indica al usuario información específica de cada archivo, detalles como el peso y la cantidad de registros que contiene. La grilla Campos indica el detalle de las columnas que el archivo Dbf contiene.

-
- Pantalla Progreso de carga

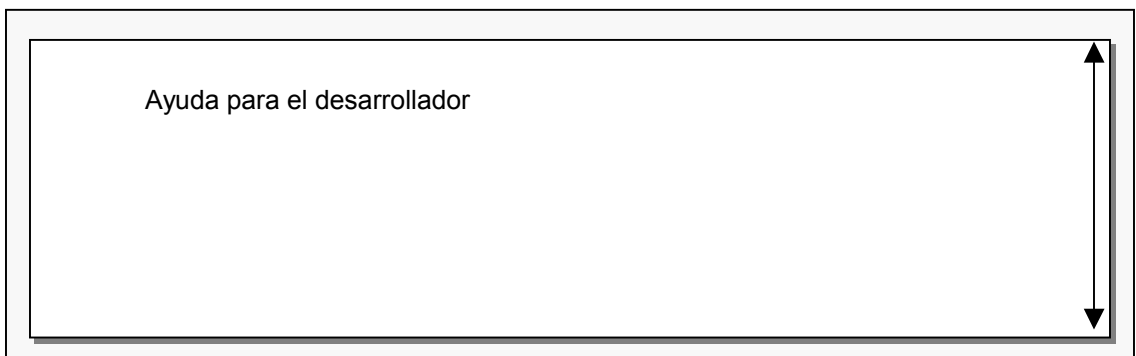
Indica en tiempo real el estado del proceso de descarga de los archivos desde el servidor remoto. El Visor esta diseñado para interactuar con mapas a través de Internet, por lo cual todos los archivos que necesite serán descargados cada vez.



(Figura 7.5.1.4) Pantalla progreso

- Pantalla de Ayuda

Muestra la ayuda orientada al desarrollador. Describe los métodos y clases usados en la construcción del Visor.

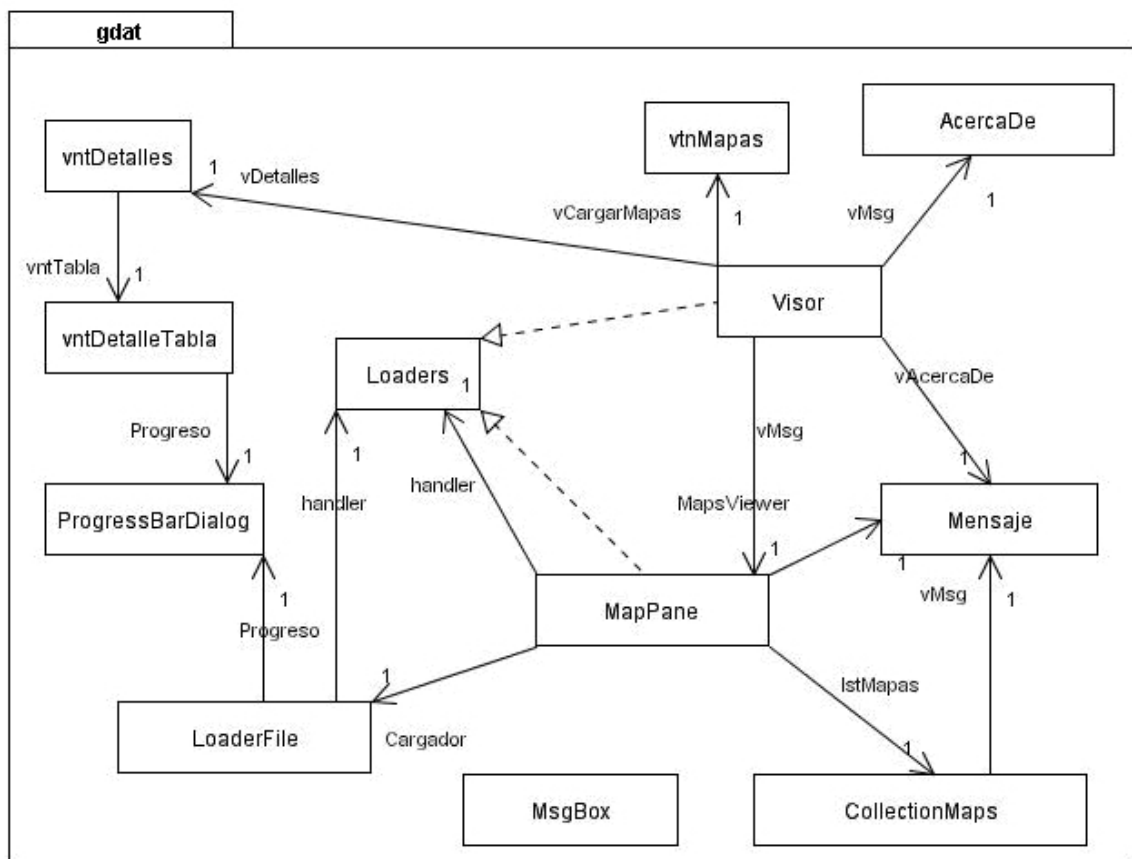


(Figura 7.5.1.5) Pantalla ayuda

7.5.2. Diagrama de clases

Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia.

El Visor esta formado por más de veinte clases, de las cuales solo se extrajo doce de las más representativas.



(Figura 7.5.2) Diagrama de clases

7.6. Implementación

La descripción de la implementación consta de lo siguiente:

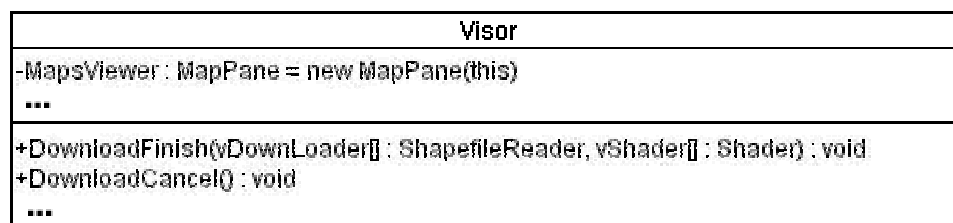
- Una descripción sobre las funcionalidades del método y su comportamiento.
- Un modelo reducido del diagrama de clases de uml, en el cual sólo se mencionan algunos de los atributos y métodos más significativos.
- Imagen representativa de la interfaz de usuario provista por el método (sólo si ésta existe).

7.6.1. Detalle de clases

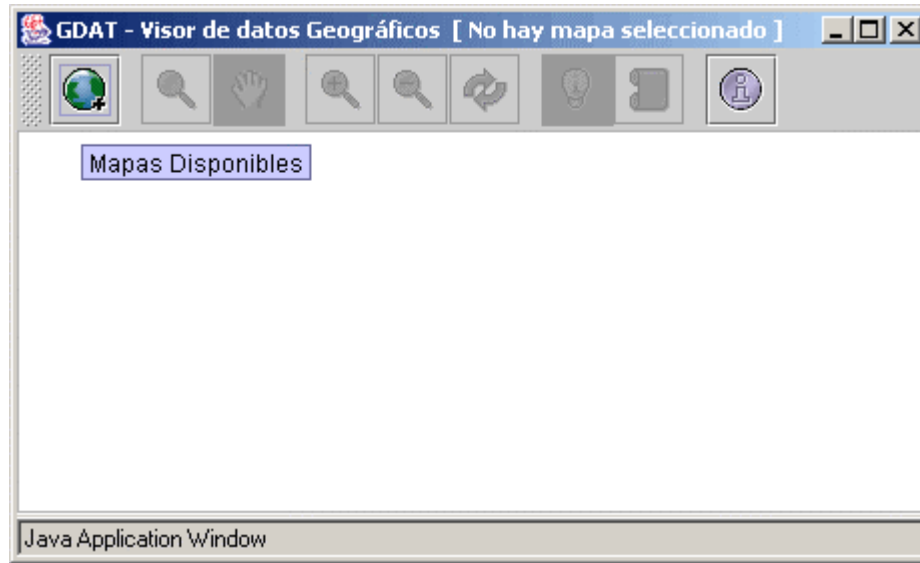
A continuación se describen partes de algunos de los métodos, contenidos en las clases correspondientes:

- Visor (JFrame)

El cual se carga al iniciar y se encarga de mostrar la interfaz de usuario en la cual se insertan los mapas.



(Figura 7.6.1.1) Clase Visor



(Figura 7.6.1.2) Formulario de la clase Visor

```
public class Visor extends JFrame implements Loaders {  
    ...  
}
```

La clase Visor implementa la interfaz Loaders que le proporciona los métodos de control del proceso de descarga, por medio de los cuales se informa si existen mapas que dibujar.

El método DownloadFinish es usado para informar si se completó la descarga y se continua con los procesos de dibujo. De lo contrario se activa el método DownloadCancel que devuelve las variables a su estado inicial.

Todos los mapas que se encuentran disponibles se descargan desde un servidor remoto ubicado en la Universidad Católica de Temuco.

Por defecto, la única opción disponible es la que permite comenzar el proceso de descarga.

Al crear los componentes del formulario se asignan los objetos contenedores, quienes realizan el proceso de dibujo y repintado de forma automática. Estos son heredados de la clase MapPane.

Primero se crea una instancia del objeto MapPane, por medio del cual se extrae el resto de los contenedores.

```
// Crea una instancia de MapPane
private MapPane MapsViewer= new MapPane(this);

// Crea el contenedor de geometrías
PanelDibujo.add(MapsViewer.getViewer(), "Center");
// Crea el contenedor de mensajes
PanelDibujo.add(MapsViewer.getLabel(), "South");

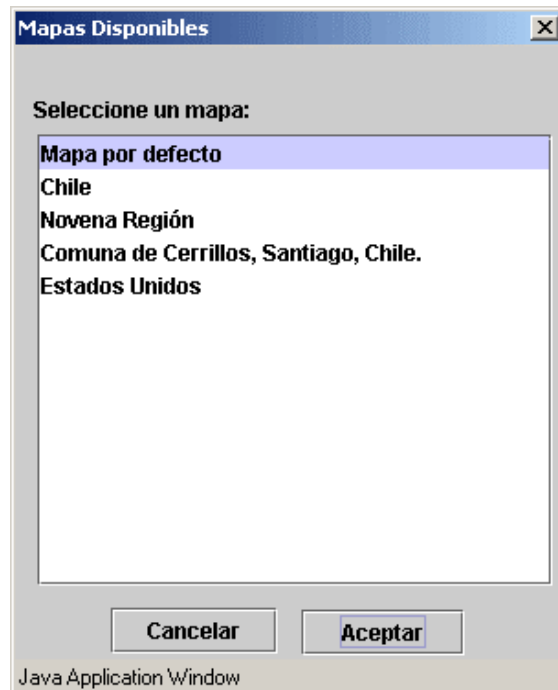
// Crea el contenedor de coberturas
ScrollCapas.getViewport().add(MapsViewer.getPaneLayers(), "Center");
// Crea el contenedor de características
PanelColores.add(MapsViewer.getTabbedPaneKeys(), "Center");
```

- vtnMapas (JDialog)

Lista los mapas disponibles en el servidor.

vtnMapas
-jList1 : JList= new JList() ...
+SetListMaps(yLista : DefaultListModel) : void ...

(Figura 7.6.1.3) Clase vtnMapas



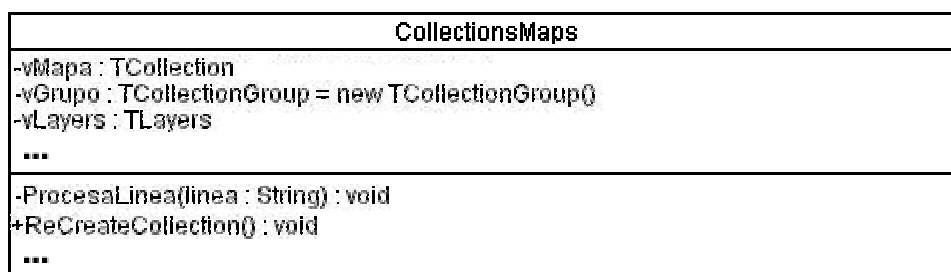
(Figura 7.6.1.4) Formulario de la clase vtnMapas

Una vez seleccionado uno de los mapas, se comienza el proceso de descarga. La lista (DefaultListModel) de mapas se obtiene a partir de MapPane quien a su vez la obtuvo de CollectionMaps.

```
public void SetListMaps(DefaultListModel vLista){  
    // remueve todos los componentes  
    jList1.removeAll();  
  
    // asigna los nuevos componentes  
    jList1.setModel(vLista);  
  
    // selecciona el primer componente de la lista  
    jList1.setSelectedIndex(0);  
}
```

-
- CollectionMaps (Class)

Está compuesta por otras tres clases (TcollectionGroup, Tcollection, TLayers), las cuales en conjunto almacenan la rutas de los mapas, nombre y otros datos pre-configurados desde un archivo maps.dat del cual se extraen. Cada vez que se instala un nuevo mapa en el servidor, debe ser registrado en este archivo para que el visor pueda acceder al mismo.



(Figura 7.6.1.5) Clase CollectionMaps

El archivo maps.dat contiene una estructura delimitada por tags, en el cual se especifican los mapas (conjunto de capas o coberturas).

Por ejemplo:

```
[MAPS]
PATH=cerrillos
NAME=Comuna de Cerrillos, Santiago, Chile.

[LAYER]
NAME=Limite Región
FILE=limite.zip
KEYS=limite.key
SHDE=use
LABL=name
[/LAYER]

[LAYER]
```

```
NAME=Solera Cerrada
FILE=solera_cerrada.zip
KEYS=solera_cerrada.key
SHDE=type
LABL=col
[/LAYER]

[/MAPS]
```

(Tabla 7.6.1) Archivo de configuración maps.dat

Este archivo es la base de todas las clases, puesto que si el archivo no existe o no se encuentra disponible, no se podrá acceder a la biblioteca de mapas.

ReCreateCollection es el método que extrae el archivo maps.dat desde el servidor para ser seccionado por ProcesaLinea quien identifica los tag y sus valores.

```
public void ReCreateCollection(){
    ...
    // abre el puerto 80 de conexión web
    URL direccion = new URL (RutaArchivo);
    URLConnection conexion = direccion.openConnection();

    // crea una entrada de archivo
    InputStreamReader stream = new InputStreamReader (conexion.getInputStream());
    BufferedReader entrada = new BufferedReader(stream);
    String linea;

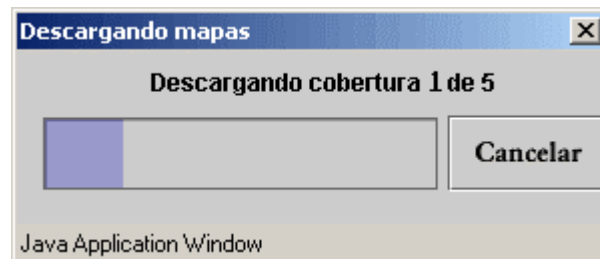
    //Recorre el archivo para lectura
    while ((linea = entrada.readLine())!=null){
        ProcesaLinea(linea);
    }
    entrada.close();
    ...
}
```

-
- ProgressDialog (JDialog)

Muestra el formulario de progreso de descarga, el cual puede ser cancelado en cualquier momento el por usuario.



(Figura 7.6.1.6) Clase ProgressDialog



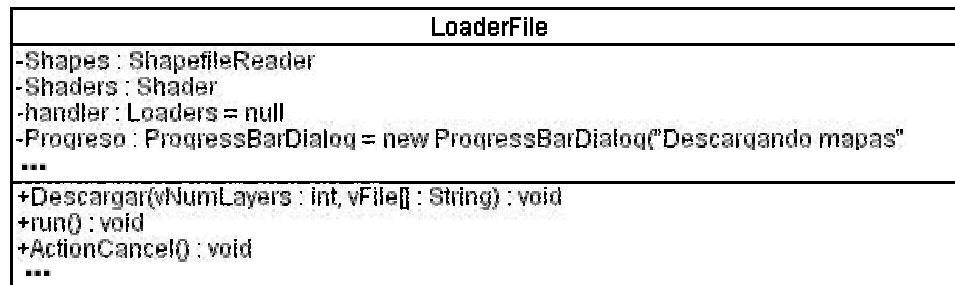
(Figura 7.6.1.7) Formulario de la clase ProgressDialog

setValue incrementa la barra de progreso, btnCancelar_actionPerformed es el método encargado de cancelar el proceso de descarga, el que se transmite para las clases que implementen la interfaz WinProgressBarEvent.

```
void btnCancelar_actionPerformed(ActionEvent e) {  
    this.setVisible(false); // oculta el formulario  
    handler.ActionCancel(); // llama al método ActionCancel contenido en la interfaz  
                             // WinProgressBarEvent  
}
```

-
- LoaderFile (Thread)

Hilo encargado de realizar la descarga de los archivos remotos. Cada cobertura puede llegar a pesar fácilmente 8MB y un mapa suele estar compuesto por varias de ellas. Pero todos los archivos se encuentran comprimidos en formato Zip, lo que reduce en más de un 50% el tamaño y tiempo de espera.



(Figura 7.6.1.7) Clase LoaderFile

Un hilo es un proceso que se ejecuta a intervalos, como un temporizador, liberando el procesador para que atienda también a otros procesos mientras se continua con la descarga de los archivos.

De esta forma se evita colgar el computador cliente durante el tiempo de descarga.

LoaderFile implementa la interfaz WinProgressBarEvent que le permite detectar si el proceso de descarga continua o si se a cancelado.

```
public class LoaderFile extends Thread implements WinProgressBarEvent {  
    ...  
}
```

El método Descargar inicializa los valores para comenzar y lanza el método Start heredado de Thread que llama a Run que se ejecuta mientras el hilo sigue vivo.

```
public void run(){
    ...
    // el ciclo se ejecuta tantas veces como coberturas tenga el mapa
    for(int i=0;i<NumLayers;i++){
        Progreso.setText("Descargando cobertura "+i+" de "+NumLayers);

        try {
            // descarga el archivo Zip que contiene el Shp y el Dbf
            Shapes[i] = new ShapefileReader(new URL(File[i]));
        } catch (MalformedURLException e) {
            Shapes[i] = null; // de no existir asigna un nulo
        }

        Progreso.setValue(i+1); // incrementa la barra de progreso
        try {
            // descarga el archivo Key que contiene el detalle de colores
            KeyFile = File[i].substring(0, File[i].length() - 4) + ".key";
            Shaders[i] = new DiscreteShader(new URL(KeyFile));
        } catch (IOException r) {
            // si no existe se asigna un color predefinido
            Shaders[i] = new MonoShader(Color.gray);
        }
    }
    ...
    handler.DownloadFinish(Shapes, Shaders);
    ...
}
```

En caso de ser cancelada la descarga por el usuario, el método ActionCancel se encarga de transmitir por medio de la interfaz Loaders el término del proceso.

```
public void ActionCancel(){
    this.stop(); // mata el hilo actual
    ...
    Progreso.setVisible(false); // oculta el formulario de progreso
    handler.DownloadCancel(); // informa al método de la interfaz Loaders
}
```

-
- Loaders (Interface)

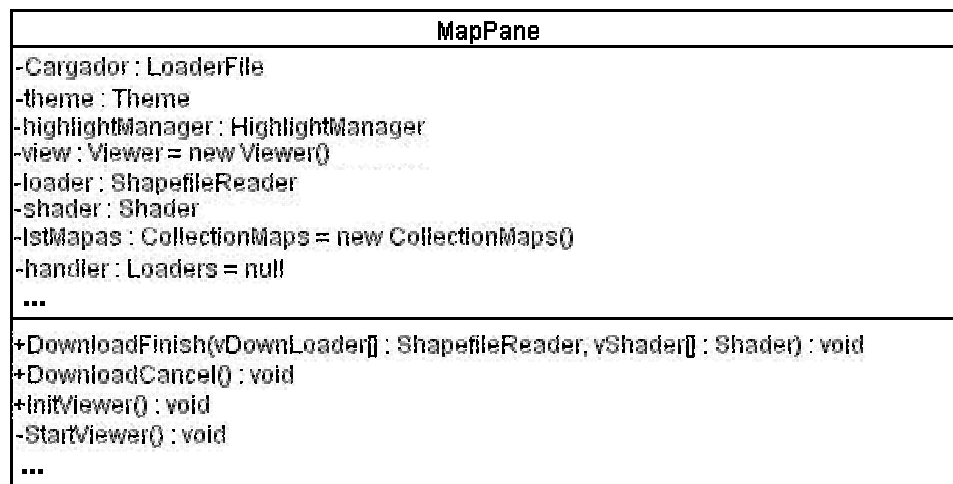
Como toda interfaz, ésta no implementa ningún método. Y es por medio de la cual se comunican los estados del proceso de descarga al resto de los objetos que la implementan.



(Figura 7.6.1.8) Clase Loaders

- MapPane (Class)

Se encarga de procesar y proveer los métodos para el dibujo de las geometrías.



(Figura 7.6.1.9) Clase MapPane

Este objeto utiliza todas las clases anteriores para lograr iniciarse. Para luego entregar las gráficas al Visor.

Cargador es una instancia de la clase LoaderFile creada en el método InitViewer que vive solo mientras se descargan los archivos, dado que es un hilo, muere cuando termina de ejecutar el método Run;

```
public void InitViewer(){  
    ...  
    // NumLayers indica el numero de coberturas  
    if(NumLayers>0){  
        // Crea los espacios necesarios para las capas disponibles.  
        // es decir, un arreglo de tamaño ajustable  
        theme = new Theme[NumLayers];  
        loader = new ShapefileReader[NumLayers];  
        shader = new DiscreteShader[NumLayers];  
        String Files[]=new String[NumLayers];  
        Cargador = new LoaderFile(this); // Instancia de LoaderFile  
        Cargador.Descargar(NumLayers,Files); // Inicia la descarga  
    }  
    ...  
}
```

DownloadFinish es lanzado cuando el proceso de descarga a concluido y comienza la inicialización de los objetos de Geotools para luego llamar a StartViewer quien es el encargado de cargar las coberturas.

El detalle de la inicialización de los objetos de Geotools1 se indica en el capítulo Nº 6, los que son usados para implementar los métodos de la interfaz Loaders.

```
public class MapPane implements Loaders{  
    ...  
}
```

- vntDetalles (JDialog)

Muestra los registros pertenecientes a cada cobertura, además de datos estadísticos.

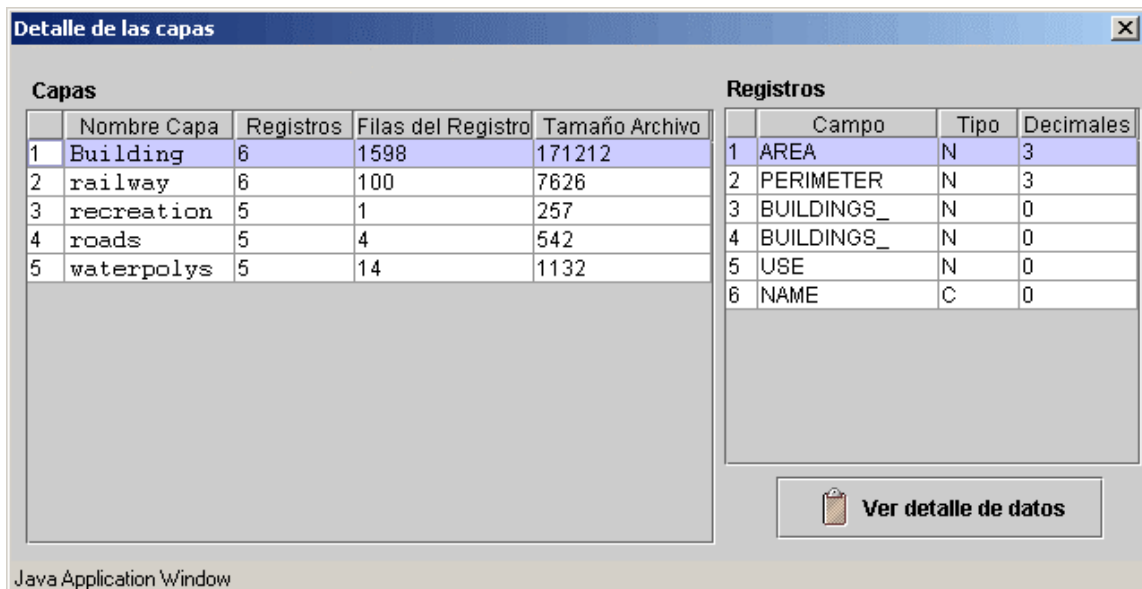
```

vntDetalles
-Shape : ShapefileReader
-TablaSHP : JTable = new JTable()
-TablaDBF : JTable = new JTable()
...
+setShapefiles(vShape[] : ShapefileReader, Max : int) : void
+MuestraDatosShape(i : int, vShape : ShapefileReader) : Vector
+MuestraDatosData(i : int, dbf : DbfFieldDef) : Vector
...

```

(Figura 7.6.1.10) Clase vntDetalles

La grilla “Registros” contiene las columnas del archivo, las que son usadas para la obtención de las filas (registros) en la clase vntDetalleTabla.



(Figura 7.6.1.11) Formulario de la clase vntDetalles

setShapefiles recibe las coberturas con las que se rellenan las tablas, MuestraDatosShape llena la tabla Capas y MuestraDatosData llena la tabla Registros.

```
public Vector MuestraDatosShape(int i,ShapefileReader vShape){
    Vector Valores=new Vector();
    String ss="";
    // número de indice
    Valores.addElement(ss.valueOf(i));

    // Nombre de la capa
    Valores.addElement(vShape.toString());

    // número de registros
    Valores.addElement(ss.valueOf(vShape.dbf.getNumFields()));

    // filas del registro
    Valores.addElement(ss.valueOf(vShape.dbf.getLastRec()));

    // Tamaño del archivo
    Valores.addElement(ss.valueOf(vShape.dbf.getFileSize()));
    ...
}
```

- VntDetalleTabla (JDialog)

Muestra el contenido completo de las coberturas, la carga de ésta puede tardar dependiendo de la cantidad de filas y columnas.

vntDetalles
-Progreso : ProgressDialog = new ProgressDialog("Llenando tabla de datos",this)
-tableDBF : JTable = new JTable()
...
+setDatos(Col : int, dbf : Dbf) : void
+ActionCancel() : void
+ActionCancel() : void
...

(Figura 7.6.1.12) Clase vntDetalleTabla

SetDatos recibe los valores del Dbf que posteriormente llenan la tabla por medio del método MuestraDatosDBF.

VntDetalleTabla implementa WinProgressBarEvent para enterarse de las acciones del usuario sobre el formulario que indica el estado.

```
public class vntDetalleTabla extends JDialog implements WinProgressBarEvent{  
    ...  
}
```

dbf.getStringCol es una matriz que contiene los registros que son insertados en la tabla, por lo que se encuentra en un bucle que incrementa las variables Col y Row.

```
public Vector MuestraDatosDBF(int Col,int Row,Dbf dbf){  
    Vector Valores=new Vector();  
    String ss="";  
    ...  
    Valores.addElement(dbf.getStringCol(Col)[Row].toString()); // valor  
    ...  
}
```

Detalle del Shape						
	AREA	PERIMETER	ROADS_	ROADS_ID	NAME	TYPE
1	704.222	156.785	2	0	Meal Hill Lane	3
2	9453.133	1911.822	3	0	Meal Hill Lane	3
3	2873.689	593.343	4	0	Surat Road	4
4	4076.821	841.660	5	0	Rock Lane	4
5	7524.931	1525.440	6	0	Clough Road	3
6	1033.101	222.165	7	0	Road	4
7	777.890	171.440	8	0	Watroyd Lane	4
8	741.306	171.610	9	0	Road	4
9	267.494	76.394	10	0	Road	4
10	928.306	263.833	11	0	London Road	4
11	5605.954	1128.916	12	0	Hawthorn Road	4
12	8070.249	1631.701	13	0	Radcliffe Road	3
13	1872.815	418.181	14	0	Road	4
14	1112.603	248.228	15	0	Road	4
15	3072.343	590.399	16	0	New North Road	4
16	1188.032	253.114	17	0	Road	4
17	2105.088	436.132	18	0	Castle Lane	4

Java Application Window

(Figura 7.6.1.13) Formulario de la clase vntDetallesTabla

7.7. Prueba

Para lograr obtener los resultados dispuestos en los objetivos, se seleccionó de entre varias alternativas algunas herramientas, de las que se destacan:

JBuilder Interprise 9:

- IDE de la empresa Borland.
- JDK 1.4.1: Paquete de desarrollo de Java.
- Geotools 1: Librerías SIG para Java.

El Visor esta disponible en tres versiones:

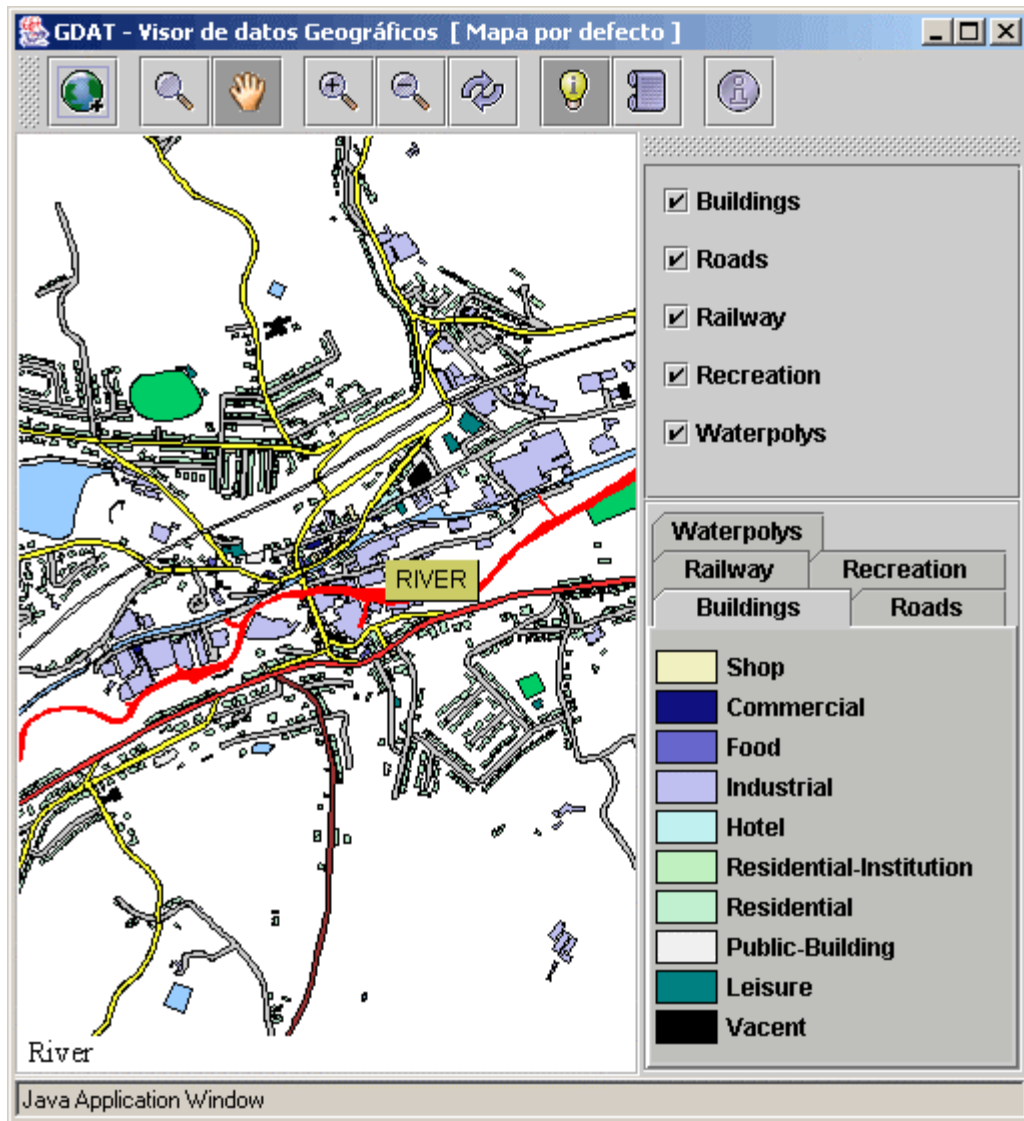
- Java Web Start: Descarga e instalación automática.
- Jar: Ejecutable empaquetado.
- Paquete de Instalación: Paquete de distribución típico de Windows.

La aplicación se encuentra actualmente instalada en el servidor dispuesto por la Universidad Católica de Temuco, en la siguiente dirección:

www.inf.uct.cl/~flara/prototipos.htm

Las pruebas realizadas han sido satisfactorias, y el proceso de instalación o ejecución del Visor solo esta delimitado por la existencia de la maquina virtual de java sobre el computador cliente.

Por lo cual, la misma página prototipos.htm es la encargada de verificar por medio de javascript, si la maquina virtual se encuentra instalada y en caso de no encontrarse se le propone al usuario instalar el paquete proporcionado por www.java.com para proseguir con la ejecución del Visor.



(Figura 7.7) Visor de mapas geográficos

8. DESARROLLOS FUTUROS

8.1. Comentarios

Lo logrado durante el transcurso de este trabajo puede ser ampliado y mejorado. Geotools1 tiene aún más funcionalidades por descubrir, que van a permitir ampliar las opciones actuales del Visualizador de datos geográficos.

Geotools2, presenta conectividad a base de datos sobre el estándar OpenGis, métodos que por razones de falta de documentación no se profundizan en este trabajo.

El Visor esta pensado para funcionar de forma genérica sobre cualquier mapa dispuesto sobre el servidor remoto. Para lo cual es necesario modificar el archivo de configuración MAPS.DAT siguiendo el mismo esquema planteado en la Tabla 7.6.1 en el que se señalan las rutas a las coberturas disponibles, además de otros datos. De esta forma se puede aumentar la cantidad de mapas disponibles para ser visualizados.

Lo que puede ser modificado a futuro para solucionar problemas puntuales, como:

- Presentar un esquema gráfico para el usuario de Internet que permita visualizar las distintas etapas de construcción inmobiliaria de una empresa cualquiera, en la cual se presenten características propias de del sitio y modelo habitacional además de la ubicación, información relevante para la toma de decisiones previa a la compra.

-
- En el ámbito Forestal son variadas las aplicaciones posibles de implementar mediante este sistema. Como por ejemplo, monitoreo remoto del avance sectorial de talado del bosque, diagramación de rutas de transporte, etc.

9. CONCLUSIÓN Y COMENTARIOS

9.1. Conclusión

Durante el transcurso de la investigación y según el manejo adquirido en las herramientas, se ha logrado implementar un Visor de datos Geográficos que permite a través de Internet visualizar los mapas dispuestos en el servidor remoto.

El software SIG se encuentra en una etapa de masificación, apoyado por proyectos de código abierto que amplían las posibilidades de implementar una solución de este tipo, en periodos de tiempo mucho más cortos.

El Visor esta construido en su totalidad por herramientas libres, lo que reduce el costo de desarrollo de aplicaciones de éste tipo, impulsando a las pequeñas empresas a adquirir software SIG.

El Visor presenta características generales que permiten a futuro ser ampliadas para resolver problemas específicos y que a su vez pretende ser una guía de iniciación para quienes incursionan en esta área, por lo cual se detallan los pasos iniciales por algunas herramientas que al final permiten construir la aplicación “Visor de datos geográficos sobre Internet”.

La mantención y ampliación de las coberturas soportadas por el Visor es configurable por medio de un archivo externo, permitiendo aumentar las posibilidades de uso del mismo.

10. BIBLIOGRAFÍA

- Apache Software Foundation - Conjunto de proyectos de software libre.
<http://www.apache.org/>
- MySQL-Hispano - Portal web, dedicado a todos los aspectos de programación y conectividad sobre mysql.
<http://www.mysql-hispano.org/>
- JavaHispano - Portal web, en el cual se pueden encontrar todos los temas posibles de realizar con java.
<http://www.javahispano.org>
- Java.Sun - Sitio oficial de Sun MicroSystem orientado a java, en el cual se puede obtener las herramientas de desarrollo.
<http://java.sun.com>
- Adictos al trabajo – Conjunto de tutoriales sobre java.
<http://www.adictosaltrabajo.com>
- Geotools - Framework de interacción SIG, conjuntos de librerías java para el desarrollo de aplicaciones bajo el estándar de OpenGis.
<http://www.geotools.org>

a. Postgresql

Linux Red Hat 9 incorpora en su distribución la versión Postgresql –7.3.2-3 por lo que no es necesario instalar otra. Para levantar los servicios, existen dos formas, una es levantarlo directamente, pero solo queda el servicio arriba hasta que se reinicie el computador.

a.1. Levantar los servicios

Para esto se ejecuta el siguiente comando:

```
# cd /etc/rc.d/init.d/postgresql start
```

El script tiene tres parámetros: start, stop, restart.

Start: inicia el servicio.

Stop: detiene el servicio.

Restart: detiene e inicia el servicio.

La primera vez que se ejecute este script se va a demorar un poco más de lo normal, porque inicializa la BD y crea una serie de archivos en el siguiente path:
`/var/lib/pgsql.`

Estos archivos son creados siempre que no existan, si el servicio ya fue levantado con anterioridad, los archivos ya fueron creados y el servicio se levanta más rápido.

De hecho es una buena opción para limpiar todas las bases de datos (por si hubiese existido algún problema), borrar todos los archivos de la ruta `/var/lib/pgsql` para luego levantar el servicio, de esta forma se crean los archivos de nuevo con las opciones originales.

Es importante señalar que si se toma esa opción se van a borrar todas las instancias de Bases de Datos que existen hasta ese momento.

La otra forma de levantar el servicio de Postgresql es que se levante como un servicio más, lo que resulta muy cómodo.

Para esto es necesario realizar dos pasos, ver en que nivel se levanta Linux y poner el script en el directorio correspondiente (o mejor dicho el link en el directorio correspondiente).

Resulta aún más fácil solo levantar el servicio para el nivel 3, nivel 4 y nivel 5. De esta forma se tiene la seguridad de que los servicios se encuentran corriendo.

Para cada nivel existe un directorio en particular en el cual se indica el orden y la cantidad de scripts que se tienen que subir. Por ejemplo el nivel 3, el directorio que contiene todos los script que se deben subir es `/etc/rc.d/rc3.d`.

Al ingresar al directorio `cd /etc/rc.d/rc3.d` y luego ejecutar `ls` se observan dos tipos de archivos, los que comienzan con una "S" y los que comienzan con una "K", seguidos por dos caracteres numéricos y luego el nombre del script a ejecutar.

Los que comiencen con una "S" son los servicios que se van a subir para el modo en cuestión. Por lo tanto si se lista el directorio /etc/rcd./rc3.d arroja algo como:

K12mysqld

K15httpd

....

....

S05kudzu

S06reconfig

S08ip6tables

....

El número indica el orden de subida de los servicios, o sea, el servicio kudzu sube primero, después el servicio reconfig y luego el servicio ip6tables. Lo lógico sería ahora encontrar un script que contenga o se llame *postgresql* y que exista como "S" o como "K".

Si el archivo existe como "K97postgresql" se debe cambiar por "S97postgresql" para que se cargue automáticamente cada vez que se inicie Linux (se hace con el comando "mv").

El número no tiene porque ser 97, podría ser cualquier otro, pero uno razonablemente alto para que otros servicios más básicos se carguen primero, por lo tanto lo mejor es cargar el script con un número alto.

Lo más probable al tratar de buscar el script (find /etc/rc.d/rc3.d -name *postgres*) es que no exista ni como “S” ni como “K”, en cuyo caso, se crea el enlace al script:

```
# cd /etc/rc.d/init.d/
# ln -s postgresql /etc/rc.d/rc3.d/S97postgresql
# ln -s postgresql /etc/rc.d/rc4.d/S97postgresql
# ln -s postgresql /etc/rc.d/rc5.d/S97postgresql
```

El proceso anterior es aún más sencillo desde un entorno gráfico como KDE, en la ventana “servicios” se selecciona el corriente y se ejecuta en los niveles deseados.

Mediante estos pasos el servicio se encuentra en ejecución y la consola (psql) de administración también, es decir, se puede crear una base de datos y usarla.

a.2. Habilitar el puerto de conexión

Paso siguiente es habilitar el puerto de conexión, por defecto el 5432. Para lo cual se edita el archivo /etc/rc.d/init.d/postgresql, en la sección o función “start” se busca la siguiente línea (o la que mas se parezca):

- Para Linux Red Hat 7.x y 8.x:

```
su -l postgres -s /bin/sh -c “/usr/bin/pg_ctl -D $PGDATA -p /usr/bin/postmaster start > /dev/null 2>&1” < /dev/null
```

Y cambiarla por:

```
su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -o -i -D $PGDATA -p /usr/bin/postmaster
start > /dev/null 2>&1" < /dev/null
```

Resumiendo solo se agregaron las opciones "-o -i".

- Para Linux Red Hat 9:

```
su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D $PGDATA -p /usr/bin/postmaster -o '-p
${PGPORT}' start > /dev/null 2>&1" < /dev/null
```

Y cambiarla por:

```
su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D $PGDATA -p /usr/bin/postmaster -o -i
start > /dev/null 2>&1" < /dev/null
```

Con este cambio hay que reiniciar el servicio nuevamente para que tenga efecto:

```
/etc/rc.d/init.d/postgresql restart
```

Si todo va bien el puerto estará listo. Para probar si el puerto está arriba se usa el comando:

```
netstat -nat
```

El cual da como resultado:

```
Proto Recv-Q Send-Q Local Address Foreign Address state
.....
Tcp    0    0 0.0.0.0:5432  0.0.0.0:*    listen
.....
```

Lo que indica que el puerto se encuentra listo.

Es importante modificar los accesos a la base de datos, en palabras simples, solo se tiene conexión desde la consola, en muchos casos se necesita habilitar

la conexión para el ip del servidor local (donde se encuentra instalado postgres) y desde otros lugares.

Para eso se edita el archivo `/var/lib/pgsql/data/pg_hba.conf` y se agregan los nuevos host (al final del archivo) que tendrán acceso a la BD.

Por ejemplo:

```
host all all 127.0.0.1 255.255.255.0 trust
host all all 197.0.0.1 255.255.255.0 trust
```

Si 127.0.0.1 y 197.0.0.1 pertenecen al mismo computador que tiene postgres, entonces lo que se hizo es permitir que todas las aplicaciones que se conecten desde esas ip's tendrán acceso.

a.3. ¿Como usar Postgresql?

Es necesario que el servidor se encuentren en ejecución, para esto se debe verificar los servicios de Linux.

Desde la consola como usuario *root* se ingresan los siguientes comando:

```
su postgres // Cuenta de postgres
createdb nombre_base_datos // Crea la base de datos
psql nombre_base_datos // Ingresa a la base de datos
```

Una vez dentro, se crean las tablas, vistas y funciones necesarias. Hay algunas vistas que son de real importancia, dado que el proceso se realiza por la consola (modo texto), será necesario automatizar algunas acciones habituales, por ejemplo:

- Todas las tablas.
- Todas las vistas.
- Todas las funciones.

Este proceso no es algo simple, Postgresql maneja la estructura de la base de datos de una forma bastante particular. Para lograr listar estos datos se llaman a tablas del sistema.

En un motor como MySQL sería suficiente ejecutar el siguiente comando para listar todas las tablas:

```
SHOW TABLES;           // Sentencias de MySQL
```

En cambio en Postgresql la misma sentencia sería así:

```
select relname from pg_class where (relkind='r') and (relname !~ '^pg_') and (relname !~ '^pga_') order by relname;
```

a.4. Automatización de Procesos Habituales

Las siguientes vistas (Views) facilitan el proceso de administración:

- Lista las funciones (Functions) existentes en la base de datos. Es importante recordar que *'nombre_base_datos'* es solo un ejemplo y debe reemplazarse por el nombre de la base de datos que se desea utilizar.

```
create view "lista_funciones" as  
select proname from pg_proc where oid>(select datlastsysoid from pg_database where  
datname='nombre_base_datos') order by proname;
```

(Tabla a.4.1) Lista las funciones

- Lista las vistas (Views) existentes en la base de datos

```
create view "lista_vistas" as  
select viewname from pg_views where (viewname !~ '^pg_') and (viewname !~  
'^pga_') order by viewname;
```

(Tabla a.4.2) Listar las vistas

- Lista las tablas (Tables) existentes en la base de datos

```
create view "lista_tablas" as  
select relname from pg_class where (relkind='r') and (relname !~ '^pg_') and (relname  
!~ '^pga_') order by relname;
```

(Tabla a.4.3) Listar las tablas

También son necesarias algunas otras sentencias

```
drop function nombre_funcion();
```

(Tabla a.4.4) Eliminar función

```
select * from pg_proc where proname='nombre_funcion';
```

(Tabla a.4.5) Listar contenido de la función

```
drop view nombre_vista;
```

(Tabla a.4.6) Eliminar vista

```
select pg_get_viewdef('nombre_vista')as vd;
```

(Tabla a.4.7) Listar contenido de la vista

```
select * from pg_user order by username;
```

(Tabla a.4.8) Listar usuarios

a.5. Procedimientos Almacenados - PI/pgsql

Como cualquier buen motor de datos, soporta funciones escritas por el usuario llamados Procedimientos Almacenados. Estas realizan acciones diversas sobre los datos según sean definidas. La novedad es que dispone de más de un lenguaje de programación para realizar el mismo proceso:

-
- PL/pgSQL SQL Procedural Language.
 - PL/Tcl Tcl Procedural Language.
 - PL/Perl Perl Procedural Language.
 - PL/Python Python Procedural Language

PL/pgSQL es el lenguaje más usado y el único que soporta SQL embebido en la función, lo que realiza el proceso mucho más fácil ya que de otra manera se necesitaría tener conocimientos avanzados sobre Perl, C o Python.

Cabe señalar que la función se ejecuta como una Transacción, la cual permite realizar un ROLLBACK en cualquier momento en caso de ocurrido un error interno o un error controlado (error intencional), es decir, deshacer todas las acciones realizadas sobre las tablas y los datos de forma de no alterar su contenido inicial.

Ejemplo de función PL/pgSQL:

```
CREATE FUNCTION get_new_folio (varchar) RETURNS integer AS '  
DECLARE  
    result folios%ROWTYPE;  
    key ALIAS FOR $1;  
BEGIN  
    LOCK TABLE folios IN EXCLUSIVE MODE;  
    SELECT INTO result * FROM folios WHERE key=fols_tipo;  
    IF NOT FOUND THEN  
        RETURN 0;  
    END IF;  
    IF result.fols_desde<result.fols_hasta THEN  
        UPDATE folios SET fols_actual=fols_actual+1 WHERE fols_tipo=key;  
        IF NOT FOUND THEN  
            RETURN 0;  
        END IF;  
    ELSE  
        RETURN 0;
```

```

        END IF;
        RETURN result.fols_actual+1;
    END;

' LANGUAGE 'plpgsql';

```

(Tabla a.5) Ejemplo de procedimientos almacenado

Detalle de las acciones realizadas por la función:

- Recibe un parámetro.
- Bloquea la tabla *folios*.
- Realiza una consulta y verifica que traiga algún resultado.
- Actualiza la tabla *folios* y verifica que los datos se guardaron de forma correcta.
- Retorna el numero de *folio*.

a.6. Pruebas

Este es un ejemplo simple que maneja solo una tabla:

Control de stock en una bodega.

```

create table stock
(
    mtrl_codigo    INTEGER        not null,
    lcls_codigo    SMALLINT       not null,
    almc_codigo    SMALLINT       not null,
    stck_disponible INTEGER        not null,
    stck_reservado INTEGER        not null,
    stck_sector    VARCHAR(10)    not null,
    constraint pk_stck primary key (mtrl_codigo, lcls_codigo, almc_codigo)
);

```

(Tabla a.6.1) Tabla Stock

```
-- Stock disponible en el almacén del local
-- Nombre: stock_disponible_almacen (integer,integer,integer);
-- Parámetros: código de material, código local, código almacen
-- Retorno OK: stock disponible
-- Retorno Error: -1

CREATE FUNCTION stock_disponible_almacen (integer,integer,integer) RETURNS
integer AS '
DECLARE
    v_material ALIAS FOR $1;
    v_local ALIAS FOR $2;
    v_almacen ALIAS FOR $3;
    v_stock integer;
BEGIN
    SELECT INTO v_stock b.stck_disponible FROM stock
    WHERE mtrl_codigo=v_material AND lcls_codigo=v_local
    AND almc_codigo=v_almacen;
    IF NOT FOUND THEN
        RETURN -1;
    END IF;

    RETURN v_stock;
END;

' LANGUAGE 'plpgsql';
```

(Tabla a.6.2) Procedimientos almacenado que verifica el stock

Para llamar a la función, solo se le debe pasar los parámetros respectivos:

```
select stock_disponible_almacen (101005,1,1);
```

(Tabla a.6.3) Llama al procedimiento de verificación de stock

b. PostGis

b.1. Instalación y Configuración

El primer paso es tener instalado el motor de datos Postgresql. La versión de Linux utilizada es Red Hat 9.

El paso siguiente es obtener los paquetes de PostGis en <http://postgis.refrations.net/download.php>. Hay varias versiones disponibles, la más reciente es la PostGis-0.8.0, no recomendada para Red Hat 9 kernel 2.4.20-8, por lo que se usará <http://postgis.refrations.net/postgis-0.7.5.tar.gz> . Paso siguiente es compilar los paquetes, para eso se necesitan algunas librerías (fuentes) que no vienen incluidas y que pertenecen a la distribución (versión) de Postgresql. Desde uno de los tantos mirrors de Postgresql se baja la versión que más se aproxime a la usada:

```
ftp://ftp.fr.postgresql.org/src/
```

Para esto es necesario obtener la versión mediante:

```
#rpm -q postgresql
```

lo que arrojará algo como "PostgreSQL -7.3.2-3".

Una vez obtenidos los paquetes correctos se procede a la instalación:

- Paso 1

Descomprimir el paquete de Postgresql en la carpeta share.

```
#cp postgresql-7.3.2.tar.gz /usr/share  
#cd /usr/share  
#tar -zxvf postgresql-7.3.2.tar.gz
```

- Paso 2

Descomprimir el paquete PostGis en la carpeta contrib.

```
#cp postgis-0.7.5.tar.gz /usr/share/postgresql-7.3.2/contrib  
#cd /usr/share/postgresql-7.3.2/contrib  
#tar -zxvf postgis-0.7.5.tar.gz
```

- Paso 3

Configurar el paquete de Postgresql pero NO se debe ejecutar el comando

make o make install.

```
#cd /usr/share/postgresql-7.3.2/  
#./configure
```

- Paso 4

Instalar PostGis.

```
#cd /usr/share/postgresql-7.3.2/contrib/postgis-0.7.5/  
#make  
#make install
```

- Paso 5

Una vez lista la instalación se debe crear una base de datos llamada “mi_bd” o usar una existente y montar PostGis sobre esta (se asume que los servicios de Postgresql se encuentra corriendo).

```
#su postgres
#createdb mi_bd
#createlang plpgsql mi_bd
```

- Paso 6

Abrir el archivo postgis.sql con algún editor y reemplazar todas las apariciones de \$libdir por /usr/local/pgsql/lib (o la carpeta donde se encuentre instalado, no donde se compilo).

```
#psql -d mi_bd -f postgis.sql
#psql -d mi_bd -f spatial_ref_sys.sql
```

Con esto ya se tiene PostGis operando en la base de datos llamada mi_bd, los detalles sobre el uso este se encuentra en el manual oficial en <http://postgis.refractor.net/postgis-spanish.pdf>. En el capítulo referente a java se especifican los códigos

b.2. Compilar PostGis.jar

Los fuentes se encuentran en el mismo paquete postgis-0.7.5.tar.gz. En el directorio postgis-0.7.5/jdbc se encuentra el archivo Makefile encargado de compilar las clases que se encuentran en la carpeta org.

Es posible que Makefile no funcione del todo bien, pero en su interior se encuentran detallados los pasos a seguir para su compilación y empaquetamiento, de la siguiente forma:

- Crear el paquete PostGis.jar

```
javac org/postgis/Geometry.java org/postgis/Point.java org/postgis/MultiPoint.java  
org/postgis/LineString.java org/postgis/MultiLineString.java  
org/postgis/LinearRing.java org/postgis/Polygon.java org/postgis/MultiPolygon.java  
org/postgis/PGgeometry.java org/postgis/PGbox3d.java
```

```
jar -cf postgis.jar org/postgis/*.java org/postgis/*.class README
```

- Crear el test: es necesario agregar el paquete PostGis.jar al CLASSPATH.

```
javac examples/Test.java  
java examples/Test
```

```
javac examples/TestServer.java  
java examples/TestServer
```

Estas clases contienen ejemplos sencillos del modo de uso, la clase Test.java debe funcionar sin mayor problema, en cambio TestServer.java es necesario editarlo para especificar el nombre del servidor y la base de datos, además de modificar la siguiente línea:

```
((org.postgresql.Connection) conn).addDataType("geometry", "org.postgis.PGgeometry");
((org.postgresql.Connection) conn).addDataType("box3d", "org.postgis.PGbox3d");
```

Que debe quedar de la siguiente forma:

```
((org.postgresql.PGConnection) conn).addDataType("geometry", "org.postgis.PGgeometry");
((org.postgresql.PGConnection) conn).addDataType("box3d", "org.postgis.PGbox3d");
```

b.2.1. Pruebas

El código siguiente muestra como realizar la conexión a Postgresql, específicamente por medio de la librería PostGis.jar e insertar y consultar datos espaciales.

PostGis.java (applet)

```
package postgistestconex;
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.sql.*;
```

```
import java.util.*;
import java.lang.*;
import org.postgis.*;
import org.postgresql.*;
import javax.swing.border.*;
```

```
public class PostGis extends JApplet {
    JPanel jPanel1 = new JPanel();
    JTextField txtServidor = new JTextField();
    JTextField txtBaseDatos = new JTextField();
```

```

JTextField txtUsuario = new JTextField();
JLabel jLabel1 = new JLabel();
JLabel jLabel2 = new JLabel();
JLabel jLabel3 = new JLabel();
JLabel jLabel4 = new JLabel();
JLabel jLabel5 = new JLabel();
JButton btnEjecutar = new JButton();
JScrollPane jScrollPane1 = new JScrollPane();
JTextArea txtResultados = new JTextArea();
JPasswordField txtPassword = new JPasswordField();
TitledBorder titledBorder1;
TitledBorder titledBorder2;
TitledBorder titledBorder3;

//Construir el applet
public PostGis() {
}
//Inicializar el applet
public void init() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
//Inicialización de componentes
private void jbInit() throws Exception {
    titledBorder1 = new TitledBorder("");
    titledBorder2 = new TitledBorder("");
    titledBorder3 = new TitledBorder("");
    this.setSize(new Dimension(519, 451));
    txtServidor.setText("197.0.0.1");
    txtServidor.setBounds(new Rectangle(12, 74, 122, 21));
    jPanel1.setLayout(null);
    txtBaseDatos.setBounds(new Rectangle(137, 74, 122, 21));
    txtBaseDatos.setText("gdat");
    txtUsuario.setText("postgres");
    txtUsuario.setBounds(new Rectangle(262, 73, 122, 21));
    jLabel1.setText("Base de Datos");
    jLabel1.setBounds(new Rectangle(138, 57, 123, 16));
    jLabel2.setBounds(new Rectangle(13, 57, 123, 16));
    jLabel2.setText("Servidor");
    jLabel3.setBounds(new Rectangle(262, 56, 123, 16));
    jLabel3.setText("Usuario");
    jLabel4.setText("Password");
    jLabel4.setBounds(new Rectangle(388, 57, 123, 16));
}

```

```

jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
jLabel5.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel5.setText("Test de conexión a Postgresql por medio de PostGis");
jLabel5.setBounds(new Rectangle(10, 5, 502, 50));
btnEjecutar.setBounds(new Rectangle(12, 107, 498, 25));
btnEjecutar.setText("Ejecutar / Conectar");
btnEjecutar.addActionListener(new PostGis_btnEjecutar_actionAdapter(this));
jScrollPane1.setBounds(new Rectangle(12, 139, 498, 302));
txtResultados.setText("");
txtPassword.setText("");
txtPassword.setBounds(new Rectangle(387, 73, 422, 21));
jPanel1.setBorder(titledBorder3);
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(txtServidor, null);
jPanel1.add(txtBaseDatos, null);
jPanel1.add(jLabel1, null);
jPanel1.add(txtUsuario, null);
jPanel1.add(jLabel3, null);
jPanel1.add(jLabel4, null);
jPanel1.add(jLabel2, null);
jPanel1.add(jLabel5, null);
jPanel1.add(btnEjecutar, null);
jPanel1.add(jScrollPane1, null);
jPanel1.add(txtPassword, null);
jScrollPane1.getViewport().add(txtResultados, null);
}

public String ValuePassword(char pass[]) {
    String Valor="";
    for(int i=0;i<pass.length;i++){
        Valor=Valor+pass[i];
    }
    return(Valor);
}

void btnEjecutar_actionPerformed(ActionEvent e) {
    Connection conn;

    String dbname = txtBaseDatos.getText();
    String dbuser = txtUsuario.getText();
    String dbpass = ValuePassword(txtPassword.getPassword());
    String dbhost = txtServidor.getText();
    String dbport = "5432";

    //Esta código crea una tabla llamada jdbc_test.
    String dbtable = "jdbc_test";

```

```

String dropSQL = "drop table " + dbtable;
String createSQL = "create table " + dbtable + " (geom geometry, id int4)";
String insertPointSQL = "insert into " + dbtable + " values ('POINT (10 10 10)',1)";
String insertPolygonSQL = "insert into " + dbtable + " values ('POLYGON ((0 0 0,0
10 0,10 10 0,10 0 0,0 0 0))',2)";

try {

    System.out.println("Creating JDBC connection...");
    txtResultados.setText("Creating JDBC connection...\n");
    Class.forName("org.postgresql.Driver");
    String url = "jdbc:postgresql://" + dbhost + ":" + dbport + "/" + dbname;
    conn = DriverManager.getConnection(url, dbuser, dbpass);
    System.out.println("Adding geometric type entries...");
    txtResultados.setText(txtResultados.getText() + "Adding geometric type
entries...\n");

    ((org.postgresql.PGConnection)conn).addDataType("geometry", "org.postgis.PGgeometr
y");

    ((org.postgresql.PGConnection)conn).addDataType("box3d", "org.postgis.PGbox3d");
    Statement s = conn.createStatement();
    System.out.println("Creating table with geometric types...");
    txtResultados.setText(txtResultados.getText() + "Creating table with geometric
types...\n");//table might not yet exist
    try {
        s.execute(dropSQL);
    } catch (Exception ep) {
        ep.printStackTrace();
        txtResultados.setText("Exception: " + ep.getMessage() + "\n");
    }
    s.execute(createSQL);
    System.out.println("Inserting point...");
    txtResultados.setText(txtResultados.getText() + "Inserting point...\n");
    s.execute(insertPointSQL);
    System.out.println("Inserting polygon...");
    txtResultados.setText(txtResultados.getText() + "Inserting polygon...\n");
    s.execute(insertPolygonSQL);
    System.out.println("Done.");
    txtResultados.setText(txtResultados.getText() + "Done.\n");
    s = conn.createStatement();
    System.out.println("Querying table...");
    txtResultados.setText(txtResultados.getText() + "Querying table...\n");
    ResultSet r = s.executeQuery("select asText(geom),id from " + dbtable);

```

```

while( r.next() )
{
    Object obj = r.getObject(1);
    int id = r.getInt(2);
    System.out.println("Row " + id + ":");
    txtResultados.setText(txtResultados.getText() + "Row " + id + "\n");
    System.out.println(obj.toString());
    txtResultados.setText(txtResultados.getText() + obj.toString() + "\n");
}
s.close();
conn.close();
}

catch( Exception ex ) {
    ex.printStackTrace();
    txtResultados.setText("Exception: " + ex.getMessage() + "\n");
}
}
}

class PostGis_btnEjecutar_actionAdapter implements java.awt.event.ActionListener {
    PostGis adaptee;

    PostGis_btnEjecutar_actionAdapter(PostGis adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.btnEjecutar_actionPerformed(e);
    }
}

```

(Tabla b.2.1.1) Prueba de código PostGis

PostGis.html

```

<html>
<head>
<title>
Conexión con PostGis
</title>

```

```
</head>
<body>
<center>
Este applet permite testear la conexión a Postgresql y utilizar PostGis
(org.postgis.*).<br><p>

<applet
  codebase = "."
  code    = "postgistestconex.PostGis.class"
  archive = "jars/test.jar,jars/pg73jdbc3.jar,jars/postgis.jar"
  name    = "PostGis"
  width   = "519"
  height  = "451"
  hspace  = "0"
  vspace  = "0"
  align   = "middle"
>
</applet>
</center>
</body>
</html>
```

(Tabla b.2.1.2) Prueba de código PostGis, página web

c. Tomcat 4.0

Tomcat es uno de los más potentes y reconocidos servidores gratuitos de Servlet del mercado. Pertenece a la familia de Apache (servidor web), es decir, es parte del proyecto Apache Jakarta Project.

<http://jakarta.apache.org/tomcat/index.html>

La instalación y configuración es bastante compleja, debido a que la documentación se encuentra en Ingles y no es lo suficientemente clara y sencilla. La documentación abarca muchos temas lo cual hace difícil diferenciar lo realmente necesario para levantar los servicios.

Existen varias versiones, la versión 3.x, la actual y estable es la 4.x, existe una versión beta 5.x. En nuestro caso solo es necesario la 4.x.

c.1. Instalación

A continuación se explica pasa a paso la instalación y configuración:

- Paso 1:

Lo primero es bajar los paquetes de Tomcat 4 del sitio oficial.

<http://apache.rediris.es/jakarta/tomcat-4/>

Para Linux existe varios tipos de paquetes disponibles como RPM, BIN y TAR.

El paquete a usar es un comprimido como jakarta-tomcat-4.1.24.tar.gz, también se realizaron pruebas con paquetes RPM pero sin tener mayores resultados.

- Paso 2:

Crear un usuario y un grupo llamado tomcat, que referencia a /usr/local/tomcat.

```
# groupadd -g 220 tomcat
# useradd -u 220 -g tomcat -c "Tomcat" -r -d /usr/local/tomcat -s "/sbin/nologin" tomcat
```

- Paso 3:

Copiar el archivo a /usr/local/ y descomprimirlo, luego borrar el archivo comprimido que ya usamos para mantener el orden en el directorio. Con esto se tendrá el directorio /usr/local/jakarta-tomcat-4.1.24 el cual se direcciona mediante el link /usr/local/tomcat, a partir de aquí cualquier referencia a Tomcat será mediante este enlace:

```
# cp jakarta-tomcat-4.1.24.tar.gz /usr/local
# tar xvzf /usr/local/jakarta-tomcat-4.1.24.tar.gz
# rm /usr/local/jakarta-tomcat-4.1.24.tar.gz
# chown -R tomcat:tomcat /usr/local/jakarta-tomcat-4.1.24
# ln -s /usr/local/jakarta-tomcat-4.1.24 /usr/local/tomcat
```

- Paso 4:

Para automatizar el proceso de carga del servidor se crea el archivo /etc/rc.d/init.d/tomcat, es importante verificar que los valores de las variables de entorno sean correctas.

```
#!/bin/sh
#
# Startup script for the Jakarta Tomcat Java Servlets and JSP server
#
# chkconfig: - 85 15
# description: Jakarta Tomcat Java Servlets and JSP server
# processname: tomcat
```

```

# pidfile: /var/run/tomcat.pid
# config:

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 0

# Set Tomcat environment.
export JAVA_HOME=/usr/local/j2sdk
export CLASSPATH=./usr/local/j2sdk/lib/tools.jar:/usr/local/j2re/lib/rt.jar
export CATALINA_HOME=/usr/local/tomcat
export CATALINA_OPTS="-Dbuild.compiler.emacs=true"
export PATH=/usr/local/j2sdk/bin:/usr/local/j2re/bin:$PATH

[ -f /usr/local/tomcat/bin/startup.sh ] || exit 0
[ -f /usr/local/tomcat/bin/shutdown.sh ] || exit 0

export PATH=$PATH:/usr/bin:/usr/local/bin

# See how we were called.
case "$1" in
  start)
    # Start daemon.
    echo -n "Starting Tomcat: "
    /usr/local/tomcat/bin/startup.sh
    RETVAL=$?
    echo
    [ $RETVAL = 0 ] && touch /var/lock/subsys/tomcat
    ;;
  stop)
    # Stop daemons.
    echo -n "Shutting down Tomcat: "
    /usr/local/tomcat/bin/shutdown.sh
    RETVAL=$?
    echo
    [ $RETVAL = 0 ] && rm -f /var/lock/subsys/tomcat
    ;;
  restart)
    $0 stop
    $0 start
    ;;
  *)
    ;;

```

```
condrestart)
    [ -e /var/lock/subsys/tomcat ] && $0 restart
    ;;
status)
    status tomcat
    ;;
*)
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
esac

exit 0
```

(Tabla c.1) Archivo de configuración de servicios de tomcat

Se le agregan los permisos correspondientes:

```
# chown root:root /etc/rc.d/init.d/tomcat
# chmod 755 /etc/rc.d/init.d/tomcat
```

Y se agrega a la lista de servicios, a partir de este momento se encuentra disponible junto con el resto de los servicios en la ventana “servicios”.

```
# chkconfig --add tomcat
# chkconfig tomcat on
```

- Paso 5:

Con todo lo hecho es suficientemente, ahora hay que levantar los servicios y corroborar que todo ha quedado bien.

Para esto se ejecuta: `/etc/rc.d/init.d/tomcat start`

Luego para probar que se encuentra corriendo visitar una de las siguientes direcciones:

http://localhost:8080/
http://127.0.0.1:8080/

c.2. Descripción del contenido del paquete

Descripción de algunos directorios y archivos antes de continuar:

- /usr/local/tomcat/bin

Contiene todos los archivos ejecutables entre ellos:

catalina.sh = Contiene la configuración de arranque.

startup.sh = Inicia los servicios.

shutdown.sh = Detiene los servicios.

- /usr/local/tomcat/conf

Contiene las configuraciones en archivos XML.

server.xml = Contiene configuraciones propias del servidor como los puertos a usar.

web.xml = Contiene las configuraciones de los directorios virtuales y su comportamiento y funcionalidades.

- /usr/local/tomcat/webapps

Contiene los archivos y páginas de los distintos directorios virtuales.

Es aquí donde están las carpetas que contienen al sitio web.

-
- `/usr/local/tomcat/webapps/ROOT`

Es el directorio por omisión. Es recomendable construir el sitio web dentro webapps y no dentro de ROOT, de forma de mantener un orden.

- `/usr/local/tomcat/work`

Contiene los archivos de trabajo, es decir, contiene las clases compiladas desde un JSP. Un jsp es una página web que contiene tags de java y se compila solo cuando se realiza su respectiva llamada por la URL, es decir, `http://localhost:8080/pagina.jsp`.

Por lo tanto la primera vez que se realice la llamada a la página el proceso será más lento porque primero debe compilarla. Eso no representa mayor problema debido a que el primero que visita la página es siempre el propio desarrollador.

- `/usr/local/tomcat/common/lib`

Contiene los archivos o librerías JAR. Tomcat carga todas las librerías que se encuentren dentro de este directorio, por lo que puede ser utilizado por ejemplo para poner los Driver JDBC, y así no tener que crear la variable entorno CLASSPATH.

c.3. Configuración

Abrir el archivo `/usr/local/tomcat/conf/web.xml` y buscar la especificación de Servlet y descomentarla, de esta forma se habilitan los servicios de Servlet para Tomcat. Debe quedar de la siguiente forma:

```
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

El servidor Tomcat por defecto atiende por el puerto 8080. También es posible utilizar Tomcat por el puerto 80 y así no depender de Apache, para esto es necesario bajar los servicios de apache para no producir un conflicto por disputa del puerto 80.

Tomcat es en si un servidor web por lo que puede asumir las funciones que realiza Apache, aunque es recomendable que Apache realice las funciones web y Tomcat solo las funciones referente a Java (Servlet, JSP).

Para esto abrir el archivo `/usr/local/tomcat/conf/server.wml` y buscar el siguiente tag y cambiar 8080 por el puerto deseado:

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8080" minProcessors="5" maxProcessors="75"
  enableLookups="true" redirectPort="8443"
  acceptCount="100" debug="0" connectionTimeout="20000"
  useURValidationHack="false" disableUploadTimeout="true" />
```

c.4. Modo de uso

Para la construcción de un sitio web dentro de webapps y que Tomcat reconozca su existencia, se realizan algunos pasos previos.

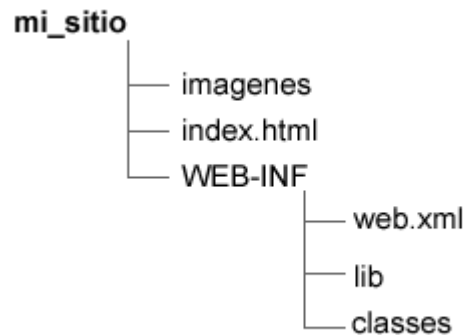
Crear un directorio dentro de webapps llamado "mi_sitio" el cual contendrá todos los archivos y subdirectorios necesarios.

Abrir `/usr/local/jtomcat/conf/server.xml` y especificar el directorio a utilizar.

Justo antes del tag `</Host>` insertar la siguiente línea:

```
<Context path="/mi_sitio" docBase="mi_sitio" crossContext="true" debug="0"
reloadable="true" trusted="false"/>
```

La estructura de archivos queda de la siguiente forma:



(Figura c.4) Directorios de trabajo de tomcat

El directorio `WEB-INF` de acceso denegado, en palabras simples, todo el contenido no es accesible. En él se encuentra el archivo de configuración `web.xml` que indica los parámetros específicos del sitio.

Configuración mínima: web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
</web-app>
```

(Tabla c.4) Configuración mínima de web.xml

El acceso a los servlet esta deshabilitado por defecto, para habilitarlos se debe especificar. Los Servlet compilados van en el directorio classes y las librerías (clases o paquetes) por seguridad es recomendable colocarlas en la carpeta lib.

Una vez concluida la configuración se reinician los servicios

```
/usr/local/tomcat/bin/shutdown.sh
/usr/local/tomcat/bin/startup.sh
```

c.5. Comunicación entre Tomcat y Apache

Tomcat es un servidor de servlet y a la vez un servidor web, pero lo recomendado es que Apache realice esa función.

La comunicación es a través del puerto 8009 habilitado por defecto al igual que el 8080.

Tres son los conectores existentes :

- mod_webapps: Desechado por su mal funcionamiento.
- mod_jk: Para Apache 1.3.x, actualmente en desarrollo.
- Mod_jk2: Para Apache 2.x, actualmente en desarrollo.

Linux Red Hat 9 incorpora en su distribución Apache 2.x, y los conectores se encuentran disponibles en el sitio oficial:

<http://apache.rediris.es/jakarta/tomcat-connectors/jk2/jakarta-tomcat-connectors-jk2-src-current.tar.gz>

Solo están disponibles los fuentes y el proceso de compilación tiene una complejidad un tanto elevada. El conector ya compilado esta disponible en http://www.inf.uct.cl/~flara/software/tomcat/jk2/mod_jk2-Apache2.zip.

Para Windows el conector esta compilado y disponibles para distintos servidores: <http://apache.rediris.es/jakarta/tomcat-connectors/jk2/binaries/win32/>

Al archivo de intercomunicación workers2.properties contiene entre otras cosas el puerto de comunicación y los directorios existen en webapps a los que Apache puede acceder, los nuevos directorios de acceso se agregan al final del archivo.

workers2.properties

```
[logger]
level=DEBUG

[config:]
file=${serverRoot}/conf/workers2.properties
debug=0
debugEnv=0

[uriMap:]
info=Maps the requests. Options: debug
debug=0

# Alternate file logger
#[logger.file:0]
#level=DEBUG
#file=${serverRoot}/logs/jk2.log

[shm:]
info=Scoreboard. Required for reconfiguration and status with multiprocess servers
file=${serverRoot}/logs/jk2.shm
size=1000000
debug=0
disabled=0

[workerEnv:]
info=Global server options
timing=1
debug=0
# Default Native Logger (apache2 or win32 )
# can be overridden to a file logger, useful
# when tracing win32 related issues
#logger=logger.file:0

[lb:lb]
info=Default load balancer.
debug=0

[lb:lb_1]
info=A second load balancer.
debug=0

[channel.socket:localhost:8009]
info=Ajp13 forwarding over socket
debug=0
tomcatId=localhost:8009
```

```
[channel.socket:localhost:8019]
```

```
info=A second tomcat instance.
```

```
debug=0
```

```
tomcatId=localhost:8019
```

```
lb_factor=1
```

```
group=lb
```

```
group=lb_1
```

```
disabled=0
```

```
[channel.un:/opt/33/work/jk2.socket]
```

```
info=A second channel connecting to localhost:8019 via unix socket
```

```
tomcatId=localhost:8019
```

```
lb_factor=1
```

```
debug=0
```

```
[channel.jni:jni]
```

```
info=The jni channel, used if tomcat is started inprocess
```

```
[status:]
```

```
info=Status worker, displays runtime informations
```

```
[vm:]
```

```
info=Parameters used to load a JVM in the server process
```

```
#JVM=C:\jdk\jre\bin\hotspot\jvm.dll
```

```
classpath=${TOMCAT_HOME}/bin/tomcat-jni.jar
```

```
classpath=${TOMCAT_HOME}/server/lib/commons-logging.jar
```

```
OPT=-Dtomcat.home=${TOMCAT_HOME}
```

```
OPT=-Dcatalina.home=${TOMCAT_HOME}
```

```
OPT=-Xmx128M
```

```
#OPT=-Djava.compiler=NONE
```

```
disabled=1
```

```
[worker.jni:onStartup]
```

```
info=Command to be executed by the VM on startup. This one will start tomcat.
```

```
class=org/apache/jk/apr/TomcatStarter
```

```
ARG=start
```

```
# For Tomcat 5 use the 'stard' for startup argument
```

```
# ARG=stard
```

```
disabled=1
```

```
stdout=${serverRoot}/logs/stdout.log
```

```
stderr=${serverRoot}/logs/stderr.log
```

```
[worker.jni:onShutdown]
```

```
info=Command to be executed by the VM on shutdown. This one will stop tomcat.
```

```
class=org/apache/jk/apr/TomcatStarter
```

```
ARG=stop
```

disabled=1

[uri:/jkstatus/*]

info=Display status information and checks the config file for changes.

group=status:

[uri:127.0.0.1:8003]

info=Example virtual host. Make sure myVirtualHost is in /etc/hosts to test it

alias=myVirtualHost:8003

[uri:127.0.0.1:8003/ex]

info=Example webapp in the virtual host. It'll go to lb_1 (i.e. localhost:8019)

context=/ex

group=lb_1

[uri:/examples]

info=Example webapp in the default context.

context=/examples

debug=0

[uri:/jsp-examples]

info=Example webapp in the default context.

context=/jsp-examples

debug=0

[uri:/examples1/*]

info=A second webapp, this time going to the second tomcat only.

group=lb_1

debug=0

[uri:/examples/servlet/*]

info=Prefix mapping

[uri:/examples/*.jsp]

info=Extension mapping

[uri:/examples/*]

info=Map the whole webapp

[uri:/jsp-examples/*.jsp]

info=Extension mapping

[uri:/jsp-examples/*]

info=Map the whole webapp

[uri:/examples/servlet/HelloW]

info=Example with debug enabled.

```
debug=10
```

(Tabla c.5) Archivo de comunicación entre Tomcat y Apache

Los archivos `mod_jk2.so`, `jkjni.so` y `workers2.properties` se copian a las siguientes carpetas de Apache.

```
# cp mod_jk2.so /usr/lib/httpd/modules
# cp jkjni.so /usr/lib/httpd/modules
# cp workers2.properties /etc/httpd/conf/
```

Luego, los módulos se cargan en el archivo de configuración `/etc/httpd/conf/httpd.conf` en la sección correspondiente y luego reiniciar apache:

```
LoadModule jk2_module modules/mod_jk2.so
```

Al seguir los pasos anteriores con seguridad el proceso funciona sin mayor problema.

Usando Tomcat de forma directa

`http://127.0.0.1:8080/examples` para versión 4.x

`http://127.0.0.1:8080/jsp-examples` para versión 5.x

Usando Tomcat a través de Apache

`http://127.0.0.1/examples` para versión 4.x

`http://127.0.0.1/jsp-examples` para versión 5.x

En caso de existir algún problema es conveniente verificar que los puertos 80, 8080 y 8009 se encuentren activos.

netstat -lnt

d. Compilación de GT2

Tres son las dependencias para lograr la compilación de este:

- Java Standard Edition versión 1.4.1 o superior.
- Java JAI.
- Apache Maven.

d.1. Java JAI

Dos son los paquetes básicos disponibles para las distintas plataformas operativas. Permiten el manejo de imágenes y renderer.

- a) Java Advanced Imaging 1.1.2
- b) Java Advanced Imaging Image I/O Tools 1.0

a) Java Advanced Imaging 1.1.2

Disponible en versión JDK y JRE para Windows, Solaris y Linux entre otros.

http://java.sun.com/products/java-media/jai/downloads/download-1_1_2.html

El paquete recomendado es jai-1_1_2-lib-windows-i586-jre.exe y se instala de forma automática sobre la carpeta en la cual se encuentra el JRE.

Luego, agregar los paquetes (JAR) al CLASSPATH.

b) Java Advanced Imaging Image I/O Tools 1.0

Disponible en formato ZIP para las distintas plataformas operativas.

<http://java.sun.com/products/java-media/jai/downloads/download-iio.html>

El paquete recomendado es `jai_imageio-1_0-lib-windows-i586.zip` y la instalación debe de realizarse de forma manual sobre la carpeta en la cual se encuentra el JRE.

Descomprimir el paquete y copiar todos los archivos JAR a `JRE\lib\ext` y todas las DLL a `JRE\bin`.

Luego, agregar los paquetes (JAR) al CLASSPATH.

d.2. Apache Maven

Es una herramienta que pertenece al proyecto Apache y permite compilar de forma sencilla grandes aplicaciones.

<http://maven.apache.org/start/download.html>

Instalar y agregar al PATH la ruta en la que se encuentran los ejecutables:

`$MAVEN_HOME/bin`

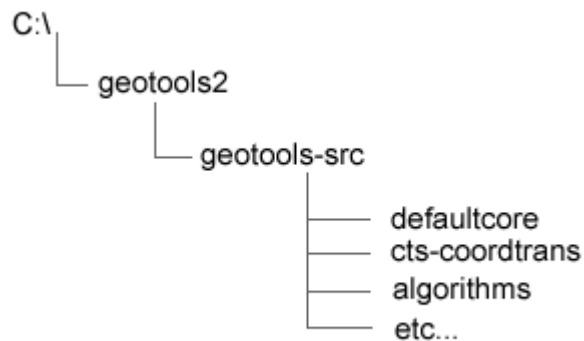
De manera opcional se puede obtener mayor información sobre el uso de esta herramienta en <http://www.devx.com/Java/Article/17204/1763/page/2> con un ejemplo sencillo.

d.3. Compilación

Bajar la última versión de los fuentes de GT2:

<http://geotools.sourceforge.net/daily/lib/gt2-src-snapshot.tar.gz>.

Y descomprimirla en una carpeta llamada geotools2, de la siguiente forma:



(Figura d.3) Estructura de directorios de geotools2

Ejecutar Maven desde la consola, dentro de la carpeta geotools2.

```
# cd geotools2
```

```
# maven build
```

El proceso de compilación puede tardar fácilmente más de una hora, dependiendo de la velocidad de conexión a Internet. Maven necesita durante el proceso, tener acceso a Internet para bajar los paquetes de dependencias (llamada repository).

Una vez realizada una compilación completa, las siguientes compilaciones tardarán mucho menos, porque los paquetes no se vuelven a bajar.

Como se muestra en la consola durante la compilación, los JAR obtenidos se copian a la carpeta repository\gt2\jars\.

También es posible descargar las fuentes ya compiladas y el repository desde

www.inf.uct.cl/~flara/software/gt2/jars

www.inf.uct.cl/~flara/software/gt2/repository.zip

e. Pruebas con Geotools

El detalle de los códigos siguientes se encuentra en el capítulo N° 7.4. También está disponible la documentación creada por medio de javadoc en www.inf.uct.cl/~flara/javadoc/.

e.1. GT1

Visor.java (applet)

```
package gdat;

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

//Geotools
import uk.ac.leeds.ccg.geotools.*;
import uk.ac.leeds.ccg.widgets.*;
import uk.ac.leeds.ccg.shapefile.*;

public class Visor extends Applet {
    //Geotools
    private Theme theme;
    private HighlightManager highlightManager;
    private Viewer view;
    private ShapefileReader loader;
    private Shader shader;
    private URL base;
    private GeoLabel label;
    private KeyHolder keys;
    private int NumLayers = 2;
    //Geotools

    //Visor Components
    private String MapaActual;
    private BorderLayout BorderLayout1 = new BorderLayout();
```

```

private JPanel PanelDibujo = new JPanel();
private BorderLayout borderLayout2 = new BorderLayout();
private TitledBorder titledBorder1;
//Visor Components

public void init() {
    base = this.getCodeBase();

    view = new Viewer();
    PanelDibujo.add(view, "Center");
    PanelDibujo.setBackground(Color.white);

    ToolBar controls = new ToolBar(view);

    String ShadeBy=null;
    String File=null;
    String Lut=null;
    String None=null;
    String Missing=null;
    String labelCol = null;

    File = "maps/buildings";
    Lut = "maps/buildings.key";
    ShadeBy = "use";
    labelCol = "name";

    Color missingColor = null;
    if(None !=null) {missingColor = Color.decode(None);}
    int missingValue=-9999;
    if(Missing !=null) {missingValue = (new Integer(Missing)).intValue();}

    try{
        loader = new ShapefileReader(new URL(base,File));
    }
    catch(MalformedURLException e){
        System.out.println("Error al cargar el archivo shapefile "+ File +": "+e);
    }

    shader=null;
    try{
        shader = new DiscreteShader(new URL(base,Lut));
    }
    catch(IOException e){
        System.err.println("Error al cargar el archivo key "+ Lut );
        shader = new MonoShader(Color.gray);
    }

```

```

    }

    shader.setMissingValueCode(missingValue);

    if(missingColor != null){shader.setMissingValueColor(missingColor);}
    theme = loader.getTheme(shader,ShadeBy);
    GeoData data = loader.readData(ShadeBy);
    shader.setRange(data);
    theme.setShader(shader);
    theme.setGeoData(data);

    label = new GeoLabel();
    highlightManager= new HighlightManager();

    theme.setTipData(data);
    theme.setHighlightManager(highlightManager);
    highlightManager.addHighlightChangedListener(label);
    label.addHighlightManager(highlightManager,loader.readData(labelCol));

    keys = new KeyHolder();
    keys.addKey(shader);

    PanelDibujo.add(keys,"East");
    PanelDibujo.add(label,"North");
    PanelDibujo.add(controls,"South");

}

public void start(){
    for(int j=0;j<NumLayers;j++){
        view.addTheme(theme);
    }
}

public Visor() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception {
    this.setLayout(borderLayout1);
}

```

```

PanelDibujo.setLayout(borderLayout2);
PanelDibujo.setBorder(titledBorder1);
this.add(PanelDibujo, BorderLayout.CENTER);

}

public static void main (String args[]) {
    Visor app=new Visor();
    app.setVisible(true);
    app.setSize(300,300);
    app.setLocation(300,300);
    app.show();
}

private void setMapaActual(String vMapa){
    MapaActual=vMapa;
}

private String getMapaActual(){
    return(MapaActual);
}

}

```

(Tabla e.1.1) Demo Geotools1 - applet

GDAT1.htm

```

<html>
<head>
<title>
    GDAT
</title>
</head>
<body bgcolor="#C8C8C8">

<center>
<table border="0" cellpadding="1" cellspacing="0" bgcolor="#4A4A4A">
<tr>
<td>
        <applet
            code = "gdat.Visor"

```

```

        archive =
"jars/geotools.jar,jars/support.jar,jars/collections.jar,jars/utilidades.jar"
        name    = "Visor"
        width   = "95%"
        height  = "95%"
        hspace  = "0"
        vspace  = "0"
        align   = "middle"
    >
</applet>
</td>
</tr>
</table>
</center>
<script>
    document.Visor.width=screen.width-50;
    document.Visor.height=screen.height-200;
</script>
</body>
</html>

```

(Tabla e.1.2) Demo Geotools1 – web

e.2. GT2

Visor.java (JFrame)

```

package gdat2;
import javax.swing.*.*;
import java.awt.*.*;
import javax.swing.border.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.io.File;
import java.net.URL;
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JMenuBar;

// Geotools 2
import org.geotools.data.shapefile.ShapefileDataSource;
import org.geotools.data.*;

```

```

import org.geotools.gui.tools.ZoomToolImpl;
import org.geotools.styling.Style;
import org.geotools.styling.StyleBuilder;
import org.geotools.map.*;
import org.geotools.util.*;
import org.geotools.ct.*;
import org.geotools.feature.*;
import org.geotools.factory.*;
// Geotools 2

// Repository
import com.vividsolutions.jts.geom.*;
import javax.vecmath.*;
import java.awt.event.*;
import org.geotools.gui.swing.*;
// Repository

public class Visor extends JFrame {

    static String AppPathResource;
    JPanel PanelVisor = new JPanel();
    BorderLayout borderLayout1 = new BorderLayout();

    public Visor() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        Visor visor = new Visor();
        visor.setLocation(300,300);
        visor.setSize(500,400);
        visor.show();
    }

    private void jbInit() throws Exception {
        AppPathResource=getClass().getResource("").toString();

        this.setLocale(java.util.Locale.getDefault());
        this.setTitle("GDAT2 - Visor de Datos Geográficos (Prueba Geotools2)");
        this.addWindowListener(new Visor_this_windowAdapter(this));
        PanelVisor.setLayout(borderLayout1);

```

```

this.getContentPane().add(PanelVisor, BorderLayout.CENTER);

GeoMaps objMaps=new GeoMaps(AppPathResource);
MapPanelImpl Mapa = objMaps.mapPane();
PanelVisor.add(Mapa);
}

void this_windowClosing(WindowEvent e) {
    System.exit(0);
}

}

class Visor_this_windowAdapter extends java.awt.event.WindowAdapter {
    Visor adaptee;

    Visor_this_windowAdapter(Visor adaptee) {
        this.adaptee = adaptee;
    }
    public void windowClosing(WindowEvent e) {
        adaptee.this_windowClosing(e);
    }
}

```

(Tabla e.2.1) Demo Geotools2 - JFrame

GeoMaps.java (class)

```

package gdat2;

import java.util.logging.Logger;
import java.awt.Dimension;
import java.awt.Color;
import java.net.URL;

// Geotools 2
import org.geotools.data.shapefile.ShapefileDataSource;
import org.geotools.data.*;
import org.geotools.gui.swing.MapPanelImpl;
import org.geotools.gui.swing.ToolMenu;
import org.geotools.styling.Style;
import org.geotools.styling.StyleBuilder;
import org.geotools.map.*;
import org.geotools.util.*;
import org.geotools.ct.*;
import org.geotools.feature.*;

```

```

import org.geotools.factory.*;
// Geotools 2

public class GeoMaps{
    private static final Logger LOGGER = Logger.getLogger("gdat2.Visor");
    private Adapters adapters = Adapters.getDefault();
    private Context context;
    private static FeatureCollection fc;
    private static Style simple;
    private static String AppPathResource;

    public GeoMaps(String AppPathResource) throws Exception {
        SetAppPathResource(AppPathResource);
        ShapefileDataSource sds = new ShapefileDataSource(new
URL(GetAppPathResource()+"maps/statepop.shp"));
        fc = sds.getFeatures();

        StyleBuilder sb = new StyleBuilder();
        simple = sb.createStyle(sb.createPolygonSymbolizer(Color.LIGHT_GRAY,
Color.BLACK, 1));
    }

    public MapPaneImpl mapPane(){
        return(createMapPane(fc,simple));
    }

    private MapPaneImpl createMapPane(FeatureCollection fc, Style style) {
        MapPaneImpl mapPane;
        Layer layer;

        ContextFactory contextFactory = ContextFactory.createFactory();

        context = contextFactory.createContext();
        layer = contextFactory.createLayer(fc, style);
        layer.setTitle("Test layer");
        context.getLayerList().addLayer(layer);
        System.out.println("Layers: " + context.getLayerList().getLayers().length);

        mapPane = new MapPaneImpl(context);
        mapPane.setBackground(Color.WHITE);
        mapPane.setPreferredSize(new Dimension(300, 300));

        return mapPane;
    }

    public void SetAppPathResource(String NewAppPathResource){
        AppPathResource=NewAppPathResource;
    }

```

```
}  
  
public String GetAppPathResource(){  
    return(AppPathResource);  
}  
  
}
```

(Tabla e.2.1) Demo Geotools2 - Class

f. Código del Visor de datos geográficos

La aplicación esta compuesta por varias clases, las cuales se indican a continuación. Para referencia y ayuda para el futuro desarrollador, en la ruta siguiente se encuentra la especificación de cada uno de los métodos de cada clase:

www.inf.uct.cl/~flara/javadoc/

f.1. Código del Visor

El código del Visor no se incluye en esta sección por razones de espacio, pero es posible obtenerlo en:

www.inf.uct.cl/~flara/fuentes.

Y en su lugar se muestra el diagrama de clases (reducido), por la misma razón anterior solo se presenta el encabezado de las clases publicas. En la siguiente dirección se encuentra el diagrama completo incluyendo todos los métodos:

www.inf.uct.cl/~flara/uml

