Capítulo 5. Implementación del sistema GISELA-X3

Este capítulo presenta cómo se construyó parte de la cartografía tridimensional, cómo se implementaron las clases de lectura para Shapefiles 3D y de XML. También se explica cómo se reconocen documentos GML y como se generan a partir de un shapefile o de una consulta a la geobase. Se muestra con detalle el uso de plantilla XSLT para transformar de un documento GML a su representación con VRML para visualizarlo. Por último se presentan una serie de interfaces desarrolladas y la descripción de su función.

5.1 Datos 3d

Para generar cartografía 3D se utilizó ArcView y dos módulos, el 3D Analyst y el Spatial Analyst [Ormsby 1999]. El 3D Analyst permite generar shapefiles en tercera dimensión, donde las geometrías pueden ser puntos, líneas o polígonos. Esta aplicación permite generar una triangulación irregular de terreno a partir de las curvas de nivel. La cartografía con la que se cuenta en el laboratorio incluye curvas de nivel cada 20 metros como se muestra en la Figura 5.1.

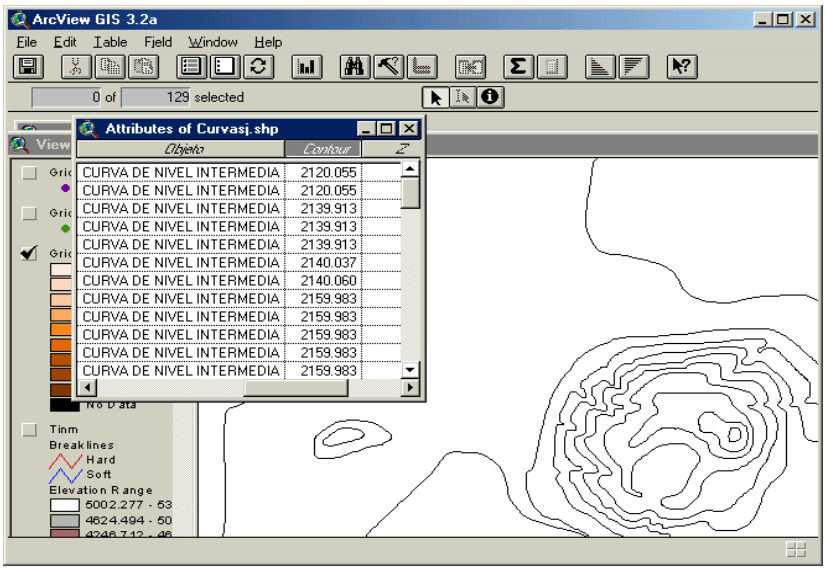


Figura 5.1 Curvas de nivel cada 20 metros.

Para la generación de un modelo digital de elevación se cargaron las curvas de nivel, y con el 3D Anayst se creo un TIN. La figura 5.2 muestra la calidad de la representación TIN.

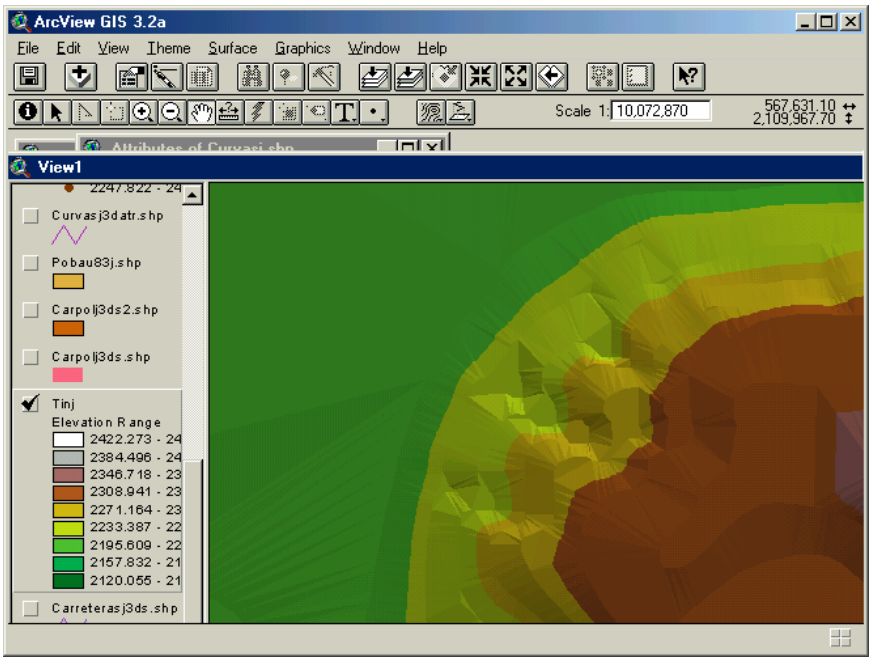


Figura 5.2 Triangulated Irregular Network

Como mencioné en el capítulo anterior ésta representación aunque muy exacta no es muy portable, por lo que de esta representación es necesario obtener un GRID. Este formato consiste en una cuadrícula regular sobre el terreno que toma la altura promedio por cada celda que ocupa. La figura 5.3 presenta el GRID generado a partir del TIN anterior. Los GRID tienen como parámetro el tamaño de las celdas, entre más pequeñas, mayor es el detalle y mayor la cantidad de información. Se probaron con distintos tamaños para determinar un valor que nos permitiera representar el terreno adecuadamente con menor espacio.

Una vez obtenido el GRID ya se tiene el modelo digital de elevación (DEM). Este nos va a permitir conocer la elevación de las entidades en 2D y generar su valor en Z. Una vez que sobreponemos la capa de líneas, puntos o polígonos al terreno (ver Figura 5.3) se transforma a un shapefile 3D.

De esta manera se transforman todas las entidades de la cartografía. Se realizaron pruebas sobre, carreteras, ríos, curvas de nivel y poblaciones. Más adelante en este capítulo se presentará como se leyeron estas capas en formato Shapefile 3D. En ArcView, se maneja el GRID en su propio formato, como no se contaba con su especificación se decidió exportar el GRID al formato Shapefile 3D de puntos. Esto se logró con un script en Avenue, que es

el lenguaje de programación de ArcView. Lo que se hace en este caso es exportar el valor de cada celda del GRID a un punto en 3D. Esta representación también nos va a permitir mas adelante reproducir el terreno en VRML más fácilmente. Lo que se obtiene en este caso es una cuadrícula o malla de puntos de elevación. La figura 5.4 muestra el resultado de esta operación.

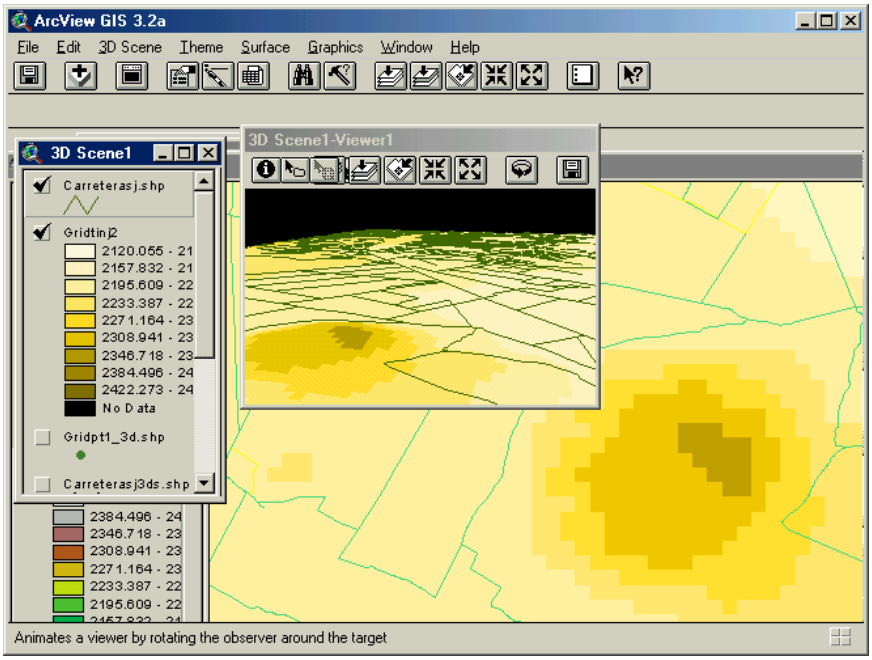


Figura 5.3 Grid generado con carreteras

5.2 Base de Datos

Este proyecto utilizó MySQL [http://www.mysql.org] como base de datos. Desarrollos anteriores trabajaron sobre Informix pero en un futuro se plantea la opción integrarse a Oracle. Por el momento se eligió MySQL por ser una base de datos estándar y de dominio público, además para las pruebas realizadas no se requerían de un gran manejador. Con MySQL se obtinene también independencia de plataforma ya que existen versiones para Windows y para Unix. Además se implementó este sistema en una máquina portátil para los usuarios potenciales y MySQL no requiere de mucho espacio ni administración compleja. Por otra parte MySQL no requiere de licencia y se encuentra disponible en Internet.

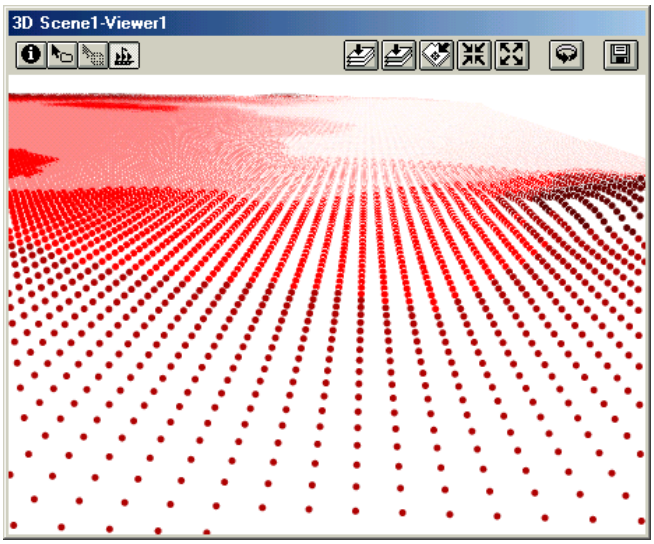


Figura 5.4 Grid transformado a puntos 3d.

Como se tenían clases de conexión a Informix con JDBC, se obtuvo el driver para MySQL y se extendieron las clases de Informix para usar MySQL con sus correspondientes tipos de datos. La diferencia más significativa fue el uso del tipo de dato BLOB en MySQL en lugar del tipo BYTE disponible en Informix para el almacenamiento binario de figuras.

5.3 Paquetes y clases

5.3.1 Lectura de archivos

El paquete de lectura de archivos contempla la lectura del formato Shapefile de ESRI. La lectura de Shapefiles con elementos en 2D fue realizada por Gómez [2000] basándose en un trabajo de García[2001]. Sin embargo fue necesario ahora incorporar la lectura de Shapefiles con elementos 3D. Para implementar esta clase se tuvo como referencia trabajo realizado anteriormente y la documentación de ESRI al respecto en su Shapefile Technical Description [http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf]

Lo primero que se lee en un archvio Shapefile es el encabezado, este contiene la longitud del archivo, la versión, el tipo de geometría que contiene y el “bounding box”. El bounding box contiene el Xmin, Ymin, Xmax, Ymax, Zmin, Zmax.. La tabla 5.1 muestra los tipos de figuras en 2D y 3D del formato Shapefile .

Tabla 5.1 Figuras del formato Shapefile

Valor	Shape Type
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ

La lectura de puntos 3d o pointZ como se denomina en el formato Shapefile, es el más sencillo. El orden de los bytes en el archivo Shapefile se muestra en la tabla 5.2:

Tabla 5.2 Lectura de PointZ

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	11	Integer	1	Little
Byte 4	X	X	Double	1	Little
Byte 12	Y	Y	Double	1	Little
Byte 20	Z	Z	Double	1	Little

La lectura de polilíneaZ y del poligonZ son muy similares, si se da uno cuenta los valores Z se incorporaron a la especificación 2D, por esta razón el valor Z del Bounding Box se

encuentra después de todos los puntos X y Y. La lectura de una polilíneaZ tiene la estructura mostrada en la tabla 5.3.

Tabla 5.3 Lectura de PolilineaZ

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	13	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y	Zmin	Zmin	Double	1	Little
Byte Y +8	Zmax	Zmax	Double	1	Little
Byte Y + 16	Zarray	Zarray	Double	NumPoints	Little
Nota: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 *NumPoints)					

El formato Shapefile es un formato binario y para la lectura de bytes se utilizó una clase ya desarrollada por Gómez[2001]. El mecanismo de lectura de estas nuevas entidades (pointz, polilinez y poligonz) se incorporó a la clase de lectura de Shapefile ya implementada. La lectura de archivos de Dbase no se modificó.

Para la lectura de archivos XML, que permitan incorporar información estructurada en este formato a la geobase se implementó un programa usando la funcionalidad proporcionada por JDOM [2001]. Esta librería permite encapsular el acceso a documentos XML usando tanto SAX como DOM (ver Capítulo 3). El diseño de esta librería permite enfocarse en el reconocimiento de las entidades XML y no en el detalle de lectura. A continuación se muestra un fragmento del programa que se encarga de obtener las coordenadas de una polilínea.

```
List featureCollection= root.getChildren("featureMember");
out.println("Este documento GML tiene "+ featureCollection.size() +" registros
                                                          featureMember:");
```

Lo primero que se hace es obtener el nodo del nodo raiz todos los nodos “featureMember”, una vez recuperados se puede conocer el número de nodos

```
   Iterator i = featureCollection.iterator();
   while (i.hasNext()) {
      Element featureMember= (Element) i.next();
      List features= featureMember.getChildren("Feature");
      out.println(" FeatureMember tiene " + features.size() + " features");
```

De cada “featureMember” se recuperan las entidades “Feature”,

```
      Iterator j = features.iterator();
      while (j.hasNext()) {
         Element feature=(Element) j.next();
         out.print("\t" + feature.getChild("name").getTextTrim() );
         List properties = feature.getChildren("property");
         out.println(" tiene  " + properties.size() + " datos descriptivos ");
         Iterator k = properties.iterator();
         while (k.hasNext()) {
                           Element property=(Element) k.next();
                           out.print("\t   Property : typeName-"+
                           property.getAttributeValue("typeName")+" type-"+
                           property.getAttributeValue("type")+" valor-"+
                           property.getTextTrim()+"\n");
        }
        String coordinates="";
        if (feature.getChild("geometricProperty").getChild("Polygon")!= null)
          coordinates= feature.getChild("geometricProperty").getChild("Polygon")
                      .getChild("outerBoundaryIs").getChild("LinearRing")
                      .getChild("coordinates").getTextTrim();
       else
          coordinates= feature.getChild("geometricProperty")
                      .getChild("LineString").getChild("coordinates")
                      .getTextTrim();
       out.print("\t geometricProperty:" + feature.getChild("geometricProperty")
              .getAttributeValue("typeName")+" coordinates->"+coordinates+"\n");
              }
        }
```

Y por cada “Feature” se recuperan sus atributos y valores, así como el “geometricProperty” y las coordenadas en caso de ser un Polígono o una polilínea. La Lectura de GML con JDOM es sencillo de implementar y permite acceder a las estructuras de los documentos de una manera estándar.

Si aplicamos este programa a un documento GML que contiene curvas de nivel en 3D tendremos la siguiente salida.

```
Este documento GML tiene 129 registros featureMember:
 FeatureMember tiene 1 features
   Curvasj3datr tiene 11 datos descriptivos
       Property : typeName-Fnode_ type-string valor-6
       Property : typeName-Tnode_ type-string valor-10
       Property : typeName-Lpoly_ type-string valor-2
       Property : typeName-Rpoly_ type-string valor-2
       Property : typeName-Length type-string valor-655.14104
       Property : typeName-Toplu83_ type-string valor-1
       Property : typeName-Toplu83_id type-string valor-66
       Property : typeName-Ids type-string valor-1002
       Property : typeName-Objeto type-string valor-CURVA DE NIVEL
                                                        MAESTRA
       Property : typeName-Contour type-string valor-2200.00500
       Property : typeName-Z type-string valor-2200
       geometricProperty:Boundary coordinates->
            568636.38118,2123088.21038,2200.00000
            568650.52800,2123059.55000,2200.00000 ....
```

Con este programa es posible obtener tanto la información descriptiva como espacial del documento GML, esta información se puede agregar posteriormente a la base de datos usando JDBC.

5.3.2 Generación de documentos GML

Para la generación de documentos GML primero se recuperan las entidades de la base de datos. Es posible estructurar en un mismo documento GML varias capas o layers con su información descriptiva y geográfica. Para generar el documento GML no se utilizó ninguna librería, únicamente se obtiene la representación GML de cada entidad con sus atributos como cadena de texto y se concatenan hasta formar todo el documento.

Una vez que se tiene el documento GML generado, éste se puede ser utilizado por el usuario. Al ser también un documento XML puede leerse desde un navegador. Un usuario experto puede almacenar este documento y utilizarlo con cualquier aplicación o inclusive puede editarlo para correcciones o modificaciones por lo que la información en este formato resulta útil.

5.4 Plantillas de traducción con XSLT

La mayoría de los usuarios contemplados no tendrán contacto con XML, por lo que es necesario traducir el documento GML a un formato que sea reconocido y útil para el usuario y sus aplicaciones. Como el documento GML es un documento XML bien formado, se puede traducir utilizando plantillas de traducción XSLT. En el desarrollo de esta tesis se definieron las siguientes plantillas de traducción:

- GML a HTML, para obtener tabulado de la información descriptiva
- GML en 2D a X3D para obtener mapas en 2D con VRML
- GML en 3D a X3D para obtener mapas en 3D con VMRL

XSLT es una especificación del W3C [www.w3c.org] y se encuentra en su versión 1.0. El objetivo de XSLT es transformar un documento en XML. XSLT es una especificación en XML e incluye componentes que reconocen tags y aplican una instrucción, ya sea obtener su información o manipularla.

Para entender como funciona una plantilla XSLT veamos un pequeño ejemplo. Si tenemos un documento GML con entidades:

```
<Feature typeName="Refugio">      <name>Secundaria No. 2</name>
       <property typeName="Teléfono">12-23-22</property>
       <property typeName="Capacidad">300</property>
       <geometricProperty typeName="Localizacion">
       <Point srsName="EPSG:UTMZ14">
       <coordinates>
                           586453.3, 2112458.7,
                           2080.4</coordinates>
              </Point>
       </geometricProperty></Feature>
```

Y si queremos transformar todos los puntos de un documento GML a una esfera en un documento X3D utilizaríamos la siguiente plantilla:

```
<xsl:template match="Point">
```

Con esta instrucción localizamos cualquier elemento “Point”,

```
	<xsl:variable name="clist" select="./coordinates"/>
	<xsl:variable name="x">
		<xsl:call-template name="transform-x">
			<xsl:with-param name="coordinate-pair" select="$clist"/>
		</xsl:call-template>
	</xsl:variable>
	<xsl:variable name="y">
		<xsl:call-template name="transform-y">
			<xsl:with-param name="coordinate-pair" select="$clist"/>
		</xsl:call-template>
	</xsl:variable>
	<xsl:variable name="z">
		<xsl:call-template name="transform-z">
			<xsl:with-param name="coordinate-pair" select="$clist"/>
		</xsl:call-template>
	</xsl:variable>
```

posteriormente seleccionamos los valores del nodo “coordinates” y los colocamos en la variable *clist*, luego llamamos a las funciones “transform-x”, “transform-y” y “transform-z” que reciben una lista como parámetro. En este caso la función transform escala y obtiene el valor de las variables “x”, “y” y “z”. Una vez listas las variables creamos un nodo *transform* de la especificación X3D y con un atributo de tipo “translation” colocamos los valores de “x”, “y” y “z”:

```
<xsl:element name="Transform" namespace="">
	<xsl:attribute name="translation">
		<xsl:value-of select="$x"/><xsl:text> </xsl:text>
		<xsl:value-of select="$y"/><xsl:text> </xsl:text>
		<xsl:value-of select="$z"/></xsl:attribute>
	<xsl:element name="Shape" namespace="">
		<xsl:element name="Appearance" namespace="">
			<xsl:element name="Material" namespace="">
			<xsl:attribute name="diffuseColor">0.8 0.0 .0</xsl:attribute>
			</xsl:element>
		</xsl:element>
		<xsl:element name="Sphere" namespace="">
			<xsl:attribute name="radius">1</xsl:attribute>
		</xsl:element>
	</xsl:element>
</xsl:element>
```

Agregamos propiedades de material y una esfera de radio 1. El resultado de aplicar la plantilla es el siguiente:

```
<Transform translation="5.864, 21.12, 0.20">
       <Shape>
              <Appearance>
                     <Material diffuseColor= 0.8 0.0 .0/>
              </Appearance>
              <Sphere radius=1/>
       </Shape>
</Transform>
```

Este es un fragmento de código de X3D, como no existen plug-ins ni aplicaciones comerciales que utilicen este formato todavía, es necesario aplicar de nuevo una plantilla que permita transformar este código a VRML. Esta plantilla, x3dtovrml97.xsl, se puede encontrar en el site de VRML [www.vrml.org].

Para la implementación de la clase Transforma (ver Capítulo 4), que se encarga de aplicar una plantilla de traducción a un documento XML, se utilizó la librería Xalan [2001] en su versión 2. Esta librería incorpora interfaces para esta tarea.

5.5 Interfaces con Servlets

Para el desarrollo de Servlets se utilizó el servidor Tomcat de Apache.org. Los criterios de selección se enfocaron, al igual que con MySQL a la portabilidad del sistema. Actualmente existen versiones tanto para Windows como para Unix de Tomcat. En el desarrollo de estas interfaces se utilizaron dos librerías; la primera del laboratorio de ICT, permite agregar componentes y estilos a las páginas en HTML mediante una clase File. La segunda librería es de la colección pública de paquetes de la editorial O'Reilly y permite recuperar archivos en formato binario a través de páginas de Internet. Los ejemplos de las interfaces desarrolladas se presentan a continuación:

Subir archivos DBF y SHP

Esta interfaz permite al usuario almacenar en la base de datos tanto archivos shapefile como archivos dbase. Para utilizar esta interfaz es necesario indicar la dirección en la máquina local de estos archivos, además se puede agregar información describiendo el contenido de los archivos y el autor. Esta interfaz puede ser utilizada entre usuarios distribuidos que necesiten compartir archivos o como un mecanismo para revisión de cartografía por parte

de un administrador. Con esta interfaz un usuario externo puede almacenar sus documentos para que un administrador los revise antes de agregarlos a la geobase.

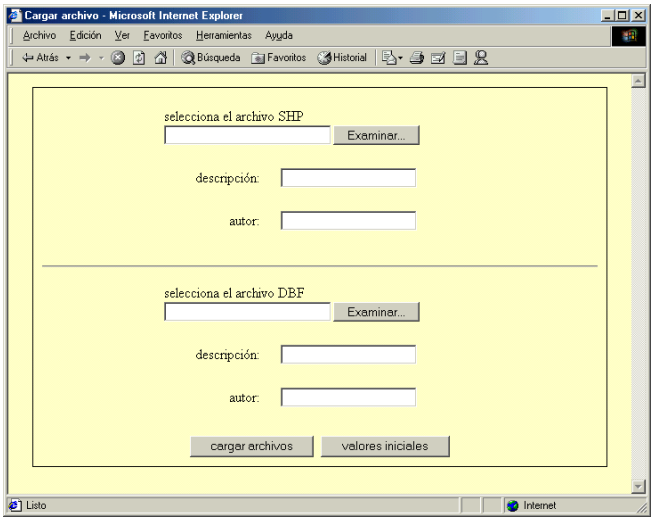


Figura 5.5 Guardando archivos SHP y DBF

Los datos proporcionados sirven a su vez de metadatos que permiten ser consultados en la interfaz de búsquedas explicada más adelante.

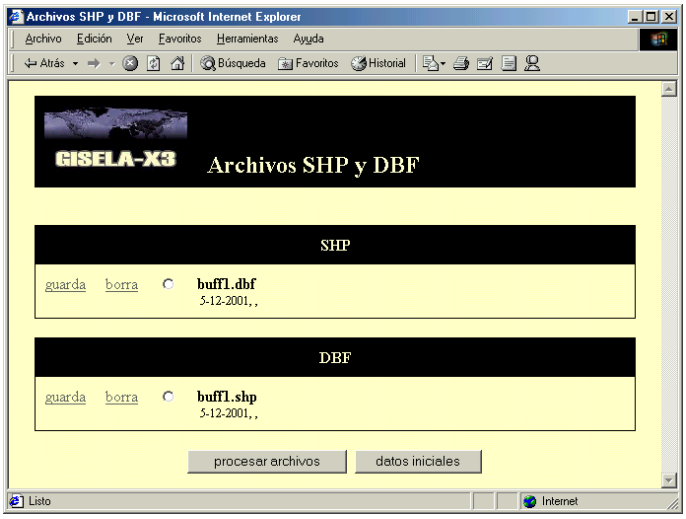


Figura 5.6 Seleccionando y procesando archivos DBF y SHP

Seleccionando y procesando archivos DBF y SHP

Esta interfaz permite seleccionar dos archivos almacenados en la base de datos, uno Shapefile y el otro Dbase para procesarlos de manera conjunta. Esta interfaz puede utilizarse para agregar la información a la geobase o para generar un documento GML de estos archivos. Este proceso lo determina el desarrollador. En esta interfaz es posible también eliminar archivos que ya no se necesitan o almacenarlos en la máquina local del cliente para su uso particular.

Procesar archivo XML

Esta interfaz nos permite seleccionar un documento XML en nuestra computadora o copiado en el área de texto y efectuar un determinado proceso. Los procesos pueden ser: almacenar en la base de datos, incorporar a la geobase, aplicar plantilla de traducción. El proceso se puede predeterminar. Esta interfaz resulta ser muy útil ya que la fuente de datos que utiliza consiste en un documento XML cualquiera. En este proyecto se utilizan documentos GML.

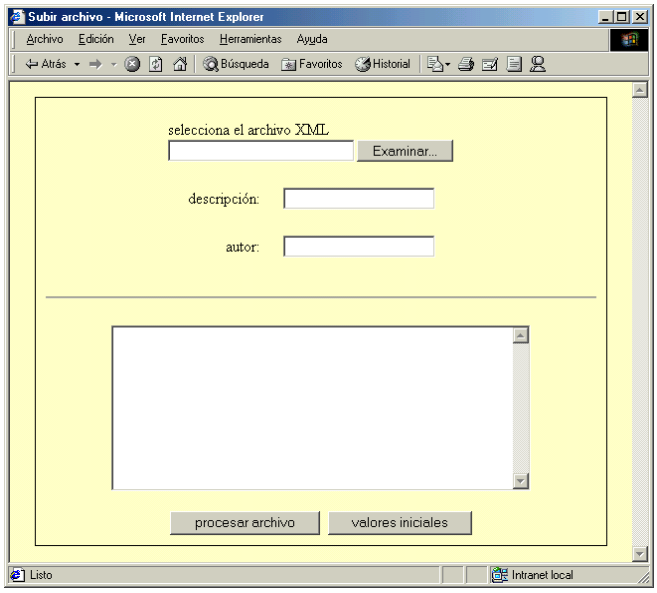


Figura 5.7 Procesar archivo XML

Generar archivo GML

Esta interfaz genera un documento GML de una o varias capas de la geobase. Lo que se necesita es marcar las capas que se requieren procesar y se obtiene como resultado un documento GML. Este documento puede ser utilizado más adelante para genrar un mapa en VRML aplicándole una plantilla de traducción (usando la interfaz anterior) o para almacenar la información en un formato conocido.

Subir archivo WRL

Esta interfaz permite agregar al catálogo de archivos de la base de datos cualquier documento VRML. Similar a la interfaz de XML, esta interfaz solicita el nombre del archivo, su descripción y el autor. Si se desea es posible también utilizar el área de texto para copiar o escribir el contenido que se requiere almacenar. La información proporcionada se utiliza en la interfaz de búsquedas. Esta interfaz se puede utilizar para almacenar mapas generados a través de una consulta y que se deseen recuperar mas adelante o para agregar componentes (símbolos, modelos o figuras) para usarse en los mapas.

Ver archivos WRL

Esta interfaz presenta una relación de todos los documentos VRML almacenados en la base de datos. Las opciones que nos presenta son:

- Guarda. Para que el usuario almacene el archivo en su computadora.
- Borra. Para que el administrador depure la información almacenada.
- Source. Muestra el contenido del archivo en un área de texto, esto es muy útil para el caso de los navegadores que interpretan siempre el contenido y no permiten ver el código original de los documentos.
- View. Para ver el contenido del archivo
- Presentación múltiple. Esta función genera un documento VRML que incluye los documentos marcados.

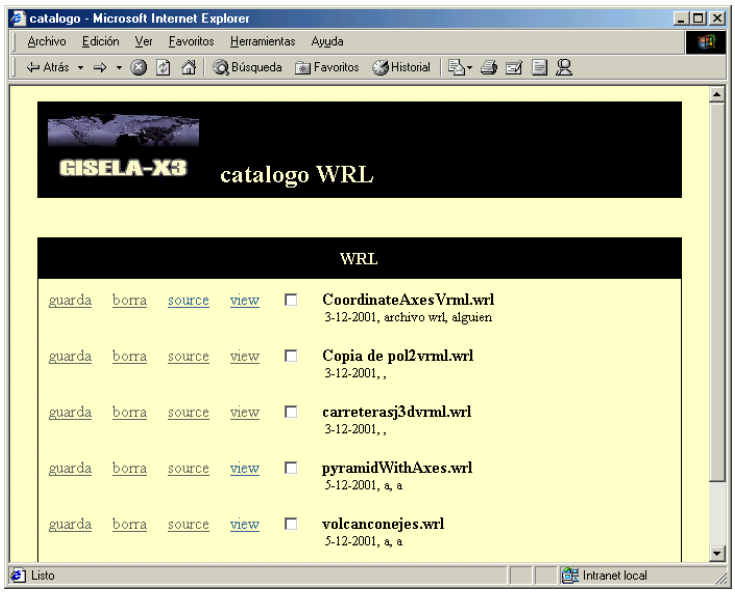
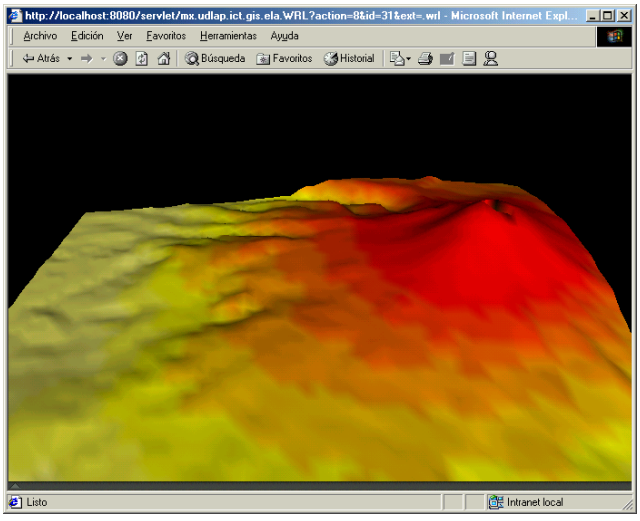


Figura 5.8 Ver archivos WRL

En este capítulo se mostró el trabajo de implementación realizado, desde lectura de archivos shapefile hasta interfaces vía Servlets pasando por lectura y procesamiento de XML. En el siguiente capítulo se presentará un caso de estudio, para demostrar el uso que se puede hacer del sistema.



Visualización de un archivo en VRML.