CAPÍTULO 6

IMPLEMENTACIÓN DEL SISTEMA GISONLINE

En este capítulo se presenta como se implementaron las partes más significativas del sistema, como el acceso y consultas a la BD por medio de ArcSDE, la transformación de los elementos a GMLv2.1.2, la transformación de documentos GMLv2.1.2 a SVG. Por último se presenta GISonline con sus diferentes funcionalidades.

6.1 CONSULTAS A LA BD

El paquete de conexión y consultas a la BD por medio de ArcSDE hace uso del API en java de ArcSDE de ESRI. Para implementar esta clase se tuvo como referencia solamente los Javadocs proporcionados por ESRI. La conexión a la BD es trivial, sin embargo la recuperación de los datos de las consultas a la BD es algo complejo sobre todo para recuperar las geometrías. Esto se debe a que ArcSDE maneja arreglos tridimensionales para manejar las geometrías la razón de esto es para facilitar las operaciones espaciales entre ellas. Maneja un arreglo donde almacena índices a las subpartes de la geometría, el segundo arreglo maneja los índices al primer punto de una subparte y el último arreglo contiene todos los puntos de la geometría. Si el arreglo de las subpartes es nulo, el arreglo de las partes contiene índices directamente al arreglo de puntos. Figuras distintas a los polígonos siempre tendrán solo una subparte. En la figura 6.1 se ve un diagrama explicando esto.

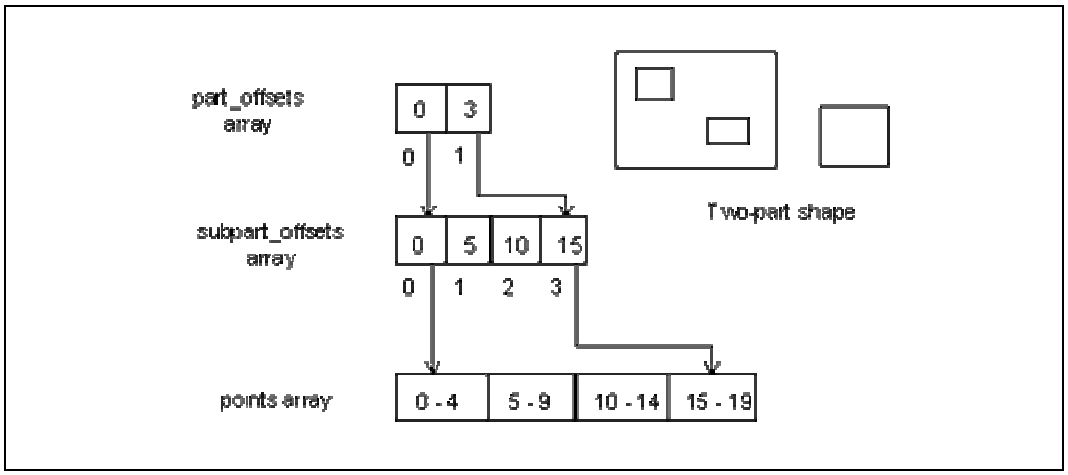

Figura 6.1 Diagrama de la estructura de un objeto SeShape

Aquí se muestra el código usado para obtener las coordenadas de una geometría a partir de SeShape:

```
double points[][][] = shape.getAllCoords();
     int numParts = shape.getNumParts();
     //Recorrido del primer arreglo
     for( int partNo = 0 ; partNo < numParts ; partNo++ ) {
       int numSubParts = shape.getNumSubParts(partNo+1);
       //Recorrido del arreglo de subpartes
       for( int subPartNo = 0 ; subPartNo < numSubParts ; subPartNo++ ) {
           int numCoords = shape.getNumPoints(partNo+1, subPartNo+1);
            //Recorrido del arreglo de puntos de la geometría
           for( int pointNo = 0 ; pointNo < numCoords*2 ; pointNo+=2) {
             Coord temp = new Coord(points[numPart][numSubPart][pointNo],
                        points[numPart][numSubPart][ (pointNo + 1)]);
           }
        }
      }
```

Cabe señalar que para realizar su conversión a GML es necesario identificar antes de hacer este proceso el tipo de geometría que contiene SeShape. Este dato es proporcionado por los tipos estáticos (identificadores) definidos en SeShape.

Para implementar la consulta espacial de BoundingBox primero construimos un SeLayer con la conexión a la base existente, el nombre de la capa a la que le

queremos aplicar el BoundingBox y el nombre de la columna espacial donde se encuentran las geometrías de la capa (SHAPE). Se obtiene el sistema de coordenadas del layer, creamos un objeto de tipo **SeExtent** con los valores X min,Ymin Xmax, Ymax (estos valores son las coordenadas del BoundigBox), y generamos un rectángulo a partir de el *extent*. Luego creamos un filtro de figuras (**SeShapeFilter**), dando los atributos del nombre de la capa, de la columna espacial, el rectángulo que se acaba de generar y se le indica el método de filtrado. En este caso aplicamos el método de búsqueda de superposición cubierta, el cual regresa los *features* que se encuentren dentro del rectángulo definido. Finalmente le asignamos el filtro espacial al *query* indicando que se desea que ArcSDE decida el mejor método de búsqueda, que no se desea que se calculen mascaras y se le pasa el arreglo de filtros. Cabe señalar que aunque solo se aplicó un solo filtro, se tuvo que guardar en un arreglo de filtros debido a la restricción en el método `setSpatialConstraints`. A continuación se muestra el código de cómo se implementó la consulta.

```
SeLayer layer = new SeLayer(this.conn, this.nombreCapa, "SHAPE");
SeShape shape = new SeShape(layer.getCoordRef());
SeExtent extent = new SeExtent(minX, minY, maxX, maxY);
shape.generateRectangle(extent);
SeShape[] shapes = new SeShape[1];
shapes[0] = shape;
SeShapeFilter filters[] = new SeShapeFilter[shapes.length];
SeShapeFilter filter = null;

for (int i = 0; i < shapes.length; i++) {
  filter = new SeShapeFilter(this.nombreCapa, "SHAPE", shapes[i],
                             SeFilter.METHOD_ENVP);
  filters[i] = filter; }
(this.query).setSpatialConstraints(SeQuery.SE_OPTIMIZE, false, filters);
(this.query). execute();
```

Con el paquete de conexión y consultas a la BD por medio de ArcSDE es posible obtener tanto la información descriptiva como espacial de las diferentes capas contenidas en la BD, está información puede ser transformada posteriormente a GMLv2.1.2 y está a su vez a SVG. Si desea obtener más información sobre este paquete puede consultar los diagramas de clases en UML y los Javadocs que se

encuentran en línea en la página del laboratorio de Geoinformación de la UDLA ("http://www.udlap.mx/~gisudla").

6.2 PAQUETE DE GENERACIÓN DE GMLV2.1.2

Sin duda alguna este es el paquete más importante de todos debido a su funcionalidad y a que varios proyectos lo utilizarán en un futuro. Como se mencionó en el capítulo anterior se modelaron los meta esquemas de GMLv2.1.2 para facilitar la reutilización y creación de nuevas clases partiendo del modelado de esquemas de aplicación. Esto se vuelve muy sencillo ya que el programador sólo tendrá que modelar sus esquemas de aplicación y crear sus clases extendiendo de las clases ya implementadas en este proyecto de tesis correspondientes a los meta esquemas mencionados anteriormente.

Para esto se implementaron 4 clases abstractas con los atributos y elementos definidos por los meta esquemas de GMLv2.1.2. La clase *Geometry* es la superclase de todas las geometrías. La clase *GeometryCollection* hereda de *Geometry* definiendo a su vez otras restricciones y es la superclase de todas las multigeometrías o colecciones de Geometrías. También se definieron las clases abstractas *AbstractFeature* y *AbstractFeatureCollection* las cuales sirven para las implementaciones de clases para nuevos esquemas de implementación. En este caso en particular las clases *GenericFeature* y *GenericFeatureCollection* heredan de estás en su implementación.

Además este paquete es independiente de las fuentes de datos ya que sus constructores son realmente muy sencillos. A continuación se muestra parte del código de uno de los constructores de la clase *Point*. Aquí el constructor recibe dos coordenadas de tipo `double` y construye el elemento *Coord* correspondiente. Para entender más el porque se define así el constructor consultar el capítulo 2 en la sección de GML.

```
/**
   * Construye un Punto con coordenas X,Y sin asignarle un GID ni un SRSName
   * @param x El valor de la coordenada en X
   * @param y El valor de la coordenada en Y
   */
  public Point(double x, double y) {
```

```
    Coord coordenadas = new Coord(x, y);
  }
```

Después para la transformación a código GMLv2.1.2 simplemente se llama al método `codificaGML()`, el cual utiliza el API de JDOM para crear un elemento punto de acuerdo a la especificación de GML y regresa un objeto de tipo **JDOM.Element**. A continuación se muestra un fragmento del método de codificación de GML para la geometría punto.

```
/**
   * Método que regresea un JDOM.Element con la representación GML del Punto
   * @return JDOM.Element Elemento con la representación GML de Punto
   */
  public Element codificaGML() {
    Element punto = new Element("Point", "gml", "http://www.opengis.net/gml");
    Element coord = new Element("coord", "gml", "http://www.opengis.net/gml");
    Element x = new Element("X", "gml", "http://www.opengis.net/gml");
    Element y = new Element("Y", "gml", "http://www.opengis.net/gml");
   coord.addContent(x.setText(Double.toString(coordenadas.getX())));
   punto.addContent(coord.addContent(y.setText(Double.toString(coordenadas.getY
   ())));
       return punto;
     }
```

Un ejemplo del resultado generado por este método es el siguiente:

```
<gml:Point >
    <gml:coord>
       <gml:X>3434323.23</gml:X> <gml:Y>5554858.12</gml:Y>
  </gml:coord>
</gml:Point>
```

Cada una de las clases del paquete sabe cual es su representación en GMLv2.1.2 con lo cual se aprovechan los recursos al construir el árbol JDOM para generar el archivo GML sólo una vez en memoria, una vez que se ha terminado de construir el documento. Esto por medio de llamadas al método `codificaGML()`, de cada una de las

clases. Es también importante mencionar que la clase principal de generación de GMLv2.1.2 *GMLDocument***,** implementa métodos para generar la salida del documento GML a un archivo, como cadena, como un elemento **JDOM.Document** o en formato comprimido (ZIP).

Si desea obtener más información sobre este paquete puede consultar los diagramas de clases en UML y los Javadocs que se encuentran en línea en la página del laboratorio de Geoinformación de la UDLA ("http://www.udlap.mx/~gisudla").

6.3 PAQUETE DE TRANSFORMACIÓN A SVG

Este paquete es el encargado de transformar un documento GMLv2.1.2 a SVG, es una modificación a las clases desarrolladas por Benjamín Rodríguez como servicio social en el laboratorio de geoinformación de la UDLAP. Esta clase se modificó para que identificara los tags de los archivos GMLv2.1.2 de GISonline y a partir de ello hiciera las transformaciones correspondientes a SVG. Para esto se hace uso de la combinación de 2 de las tecnologías mencionadas en el capítulo 3 SAX y JDOM. SAX se usa para hacer un análisis línea por línea de la información del archivo GML y JDOM se usa para la creación del archivo de SVG. A continuación se muestra un fragmento del código modificado para la conversión de los archivos GML de GISonline a SVG. Aquí se identifica el *tag featureMember* del Namespace "gml:" y por cada *featureMember* se obtienen los *GenericFeatures*. Después por cada *GenericFeature* se identifica el tipo de geometría obteniendo sus coordenadas y creando el archivo SVG.

```
java.util.List featureMembers = root.getChildren("featureMember", nsgml);
 Iterator i = featureMembers.iterator();
 while (i.hasNext()) {
  Element featureMember = (Element) i.next();
  java.util.List genericFeatures = featureMember.getChildren ("GenericFeature");
```

Cada uno de los paquetes del Modelo fueron programados con el fin de que fueran componentes reutilizables. Primero se realizaron pruebas de desempeño en consola. Después se integraron de manera sencilla a la aplicación y se realizaron otras pruebas

de desempeño. Los resultados de estas pruebas se mencionarán en el capítulo siguiente.

A continuación se describirán las clases que se implementaron para manejar los formularios y acciones del controlador. Si se desea tener información de cómo se implementaron los archivos de internacionalización de mensajes, la configuración del archivo de configuración y el uso de las librerías de *tags* de Struts favor de consultar el Apéndice D.

6.4 Implementación del Controlador y la Vista

Como se mencionó en el capítulo anterior se manejarán 2 beans de formulario y 3 beans de acción. La implementación de los *beans* de formulario es muy sencilla. Sólo se definen métodos `get()` y `set()` por cada campo del formulario, un método *reset()* para limpiar la información del formulario y un método `validate()` que valida la información introducida en los campos. El método `validate()` regresa un conjunto de **ActionErrors** indicando las entradas que no fueron válidas, para que el usuario las corrija. Un ejemplo de esto se puede ver en el Apéndice D.

Para la implementación de los *beans* de acción se crearon 3 clases. La clase *ConexionAction* la cual se encarga de realizar validar en conjunto con *ConexionForm* la conexión a la base de datos y en caso de que se pueda realizar:

- Realiza la consulta de todas las capas disponibles en la BD.
- Guarda el nombre de las capas disponibles en un objeto de sesión.
- Guarda los datos de conexión a la BD.
- Cierra la conexión a la BD.
- Redirecciona a la siguiente Vista donde se muestran todas las capas disponibles a consultar en la BD.

En la Figura 6.2 se muestran las pantallas correspondientes a la conexión y consulta de capas disponibles en la BD. En la pantalla de la izquierda se muestra el comportamiento del sistema en caso de que el usuario introduzca una entrada no válida o que no se pueda establecer la conexión a la BD. En la pantalla de la derecha se observa el comportamiento en caso de establecer la conexión a la BD mostrando las capas disponibles a consultar.

Figura 6.2 Vistas para la Conexión a la BD

La clase GISAction es la encargada de realizar todas las operaciones GIS de la aplicación, está clase se ocupa de interactuar con cada uno de los componentes del modelo para realizar las siguientes funcionalidades:

- Consulta de los meta datos de la capa seleccionada.
- Consulta espacial aplicando un BoundingBox a una capa.
- Transformación de los resultados de las consultas a GMLv2.1.2.
- Visualización del documento GMLv2.1.2 en pantalla.
- Transformación del documento GMLv2.1.2 a SVG.
- Visualización del documento SVG generado.
- Descarga del archivo GMLv2.1.2 en formato comprimido.

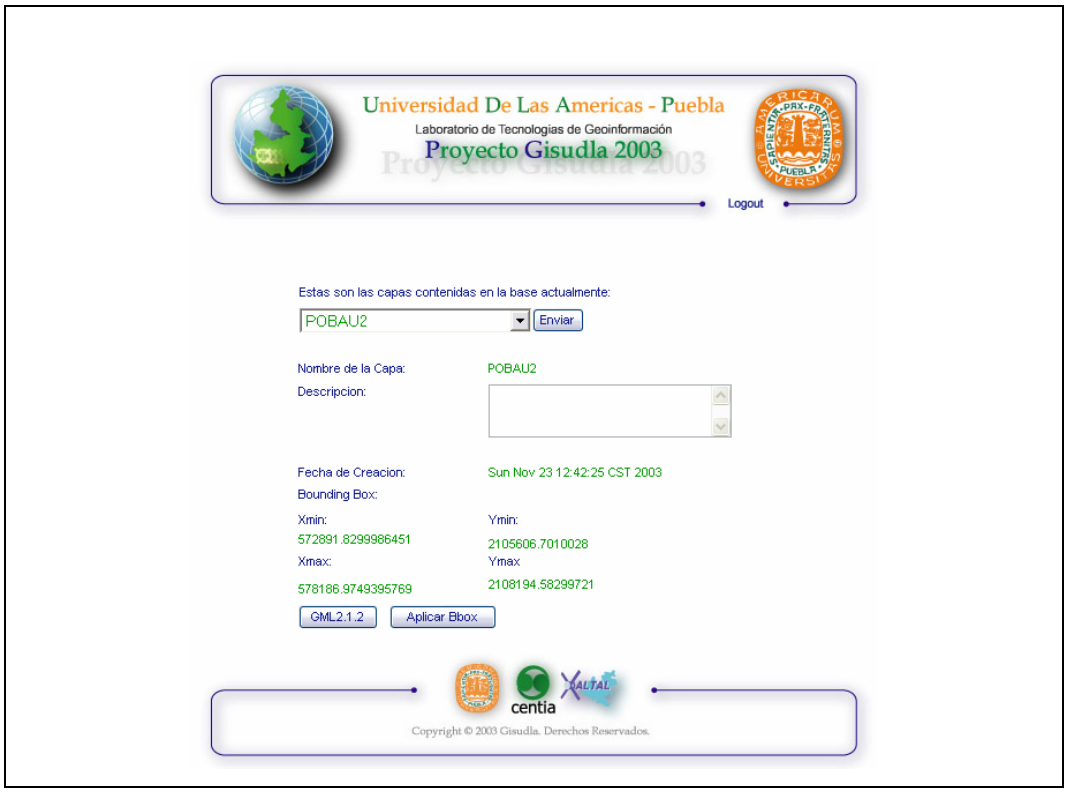

Figura 6.3 Consulta de los meta datos de una capa

Todas estas vistas se generan aparentemente sobre un solo JSP, sin embargo no es así. Por cada evento solicitado por el usuario en GIS.jsp, la clase *GISAction* procesa la información del evento y reenvía al GIS.jsp que hace uso del template de la aplicación, con los parámetros correspondientes, así el *template* visualiza solo las JSP's relacionadas al evento. A continuación se muestra parte del código de la clase **Action** para manejar la petición de ver el código del documento GML generado:

```
//Cuando se solicita ver el código GML generado de la capa en el navegador
    //se redirecciona a GIS.JSP activando el atributo gmlCode
    if (action.equals("gmlCode")) {
      session.setAttribute("activar", "gmlCode");
      return (mapping.findForward("success"));
    }
```

Este es el fragmento de la JSP GIS.jsp para procesar la vista a partir del atributo gmlCode. Aquí se hace uso de las *tag libraries* de Struts indicando que jsp incluir sobre cada celda definida en el template. La parte que esta dentro de *logic* analiza si el

elemento "activar" tiene definido el atributo "gmlCode" y como esto es verdadero incluye también esta JSP en la vista a generar (ver figura 6.4).

```
…
<template:put name='encabezado' content='/encabezadosw.jsp' />
<template:put name='contenido1' content='/queryCapas.jsp'/>
<logic:equal name="activar" value="gmlCode">
   <template:put name='contenido2' content='/querydocGML.jsp'/>
 </logic:equal>
<template:put name='pie' content='/pie.jsp' />
…
```

En la figura 6.5 se puede observar la vista generada para la visualización del archivo SVG generado a partir de este documento. Es importante señalar que para la implementación de la vista se incluyeron las normas de accesibilidad correspondientes a la visualización dentro de los mensajes internacionalizados, así el texto alternativo de las imágenes (atributo "alt") se muestra en inglés o en español según sea el caso.

Figura 6.4 Vista generada para la presentación del código GML

Finalmente la clase *LogoutAction* es la encargada de cerrar la sesión del usuario, eliminar todos los objetos creados durante la sesión y los archivos temporales creados para que el usuario pudiera descargar los archivos GML comprimidos y los archivos SVG. En un futuro se piensa corregir esto y generar los archivos por medio de un buffer. A continuación se muestra un fragmento de esta clase en el cual se muestra el procedimiento para borrar los archivos temporales generados en la sesión del usuario, los objetos usados en la sesión y finalmente cierra la sesión y redirecciona a la página principal de la aplicación.

```
if (session.getAttribute("zip") != null || session.getAttribute("svg") != null)
{
      File zip = new File(servlet.getServletContext().getRealPath("") +
                          System.getProperty("file.separator") + "gml"
                          + session.getId() +
                          System.getProperty("file.separator") + ".");
      File files[] = zip.listFiles();

      for (int i = 0; i < files.length; i++) {
        System.out.println("Deleting: " + files[i]);
        files[i].delete();
      }
      zip.delete();
   }
    session.removeAttribute("MiAtributo");
    …..
    session.invalidate(); // cierra la sesión
    //   Redirecciona  a la página inicial (“succes esta definido en el archivo
    //de configuración de struts”)
    return (mapping.findForward("success"));
```

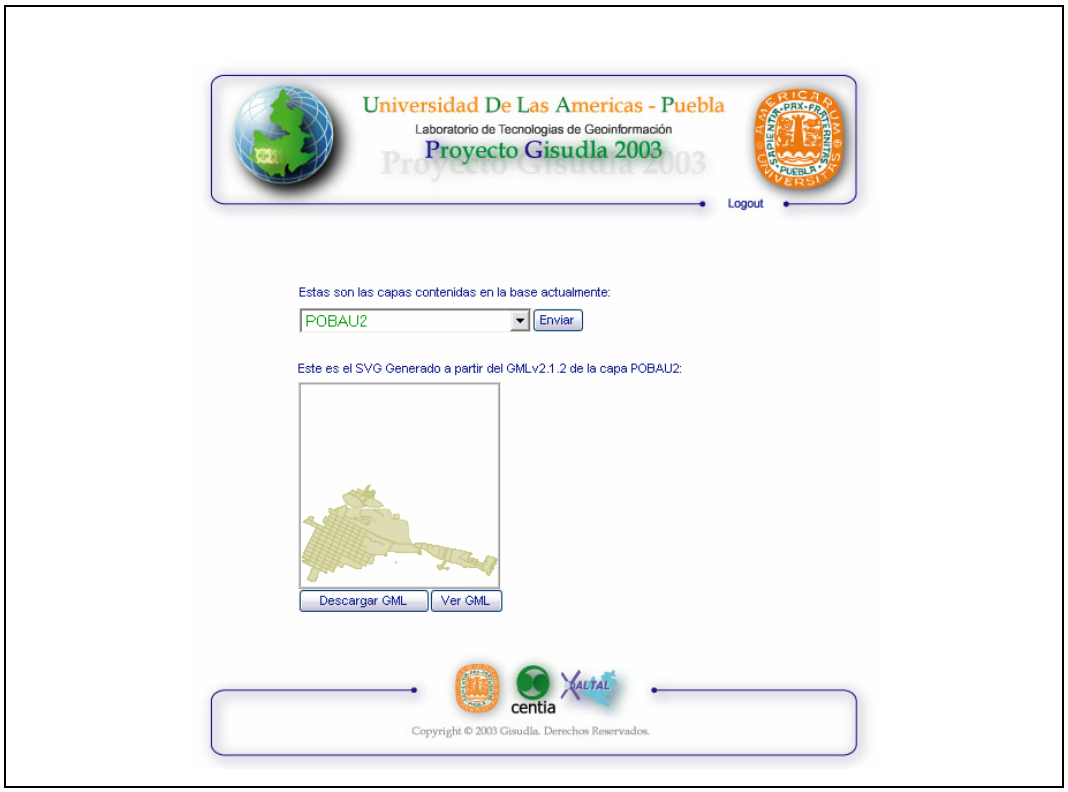

Figura 6.5 Vista generada para la presentación gráfica (SVG), del documento GML

Al concluir la implementación de GISonline y hacer las pruebas correspondientes se decidió crear un archivo de tipo ".war" de la aplicación para que se pueda auto-instalar de una manera sencilla sobre el servidor de páginas dinámicas Apache Tomcat. En el siguiente capítulo se mencionaran los resultados de las pruebas realizadas a este sistema, las conclusiones y el trabajo a futuro.