Capitulo 6

Implementación

En este capitulo mostraremos la forma en que realizamos nuestro sistema, como son los accesos a disco para leer los documentos GML y SVG, además de la aplicación de las hojas de estilo y sin mencionar la creación de documentos GML. En capítulos anteriores hemos discutido sobre las tecnologías de XML como SVG, XSLT, GML. En este capitulo presentaremos la implementación de nuestro sistema apoyado en estas herramientas. Se describirán también los problemas y las limitaciones que se fueron encontrando con el desarrollo del sistema.

6.1 Metodología de la implementación

Uno de los más importantes objetivos de nuestro sistema es la separación y compartimiento de información. Las hojas estilos así como las creaciones de archivos GML y SVG son ejemplos de esta separación sin nombrar las separaciones de los modelos, controladores y las vistas. Esta forma de separamiento de información nos provee de un fácil manejo y modificación de documentos, reduciendo la complejidad de la programación así como el uso de hojas de estilo y la visualización de documentos SVG. Los datos GML tienen que ser validados para el aseguramiento de la estandarización mediante aplicación de un schema. La salida del documento SVG que nos genera las hojas de estilo tiene que ir en concordancia con la especificación de SVG 1.0 la cual es soportada por todos los visualizadores en el tiempo de la implementación.

Editix fue usado para la generación de documentos GML así como el de los schemas y hojas de estilo. A continuación en las siguientes secciones mostraremos paso a paso como se fueron implementando todas estas tecnologías.

6.2 Documentos GML

La información que fue usada en este sistema ha sido basada en simulación de fenómenos de tráfico, del centro de investigación alemán Fraunhofer FIRST, además de la colaboración con el laboratorio de Geotecnologías del CENTIA de la UDLA, Puebla.

A continuación en la figura 6.1 se muestra un diagrama general de la implementación de nuestro software.

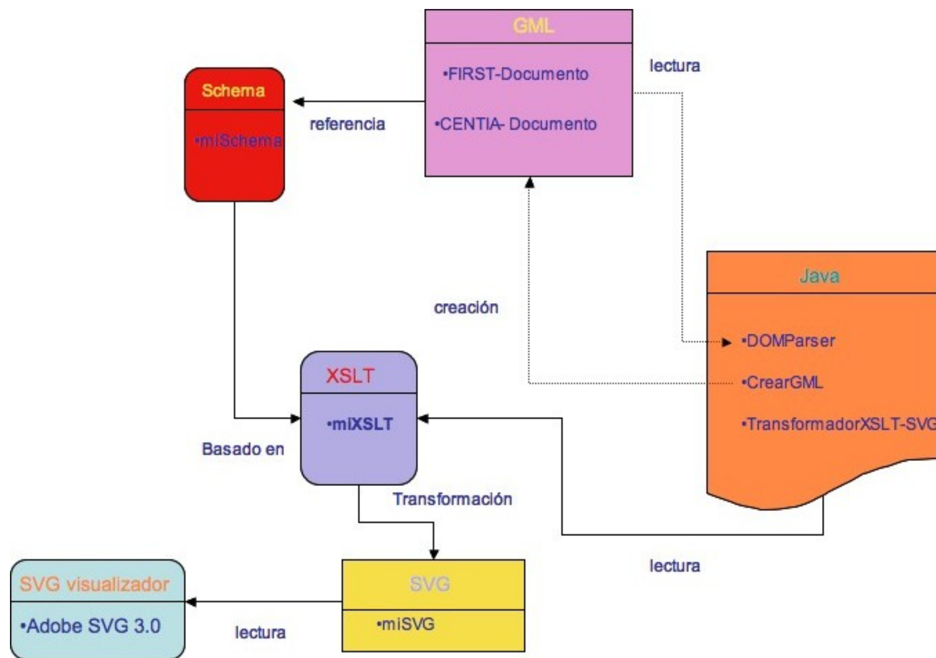

Figura 6.1 diagrama de la implementación de nuestro sistema

De acuerdo a nuestro modelo de datos, tenemos 2 elementos bases como lo son *stretch* y *car*, y estos tienen 2 características cada uno las cuales son densidad, km para *stretch* y tiempo y velocidad para *car*.

Nuestros conjuntos de datos GML consisten en 2 documentos FIRST, CENTIA. Todos estos datos fueron validados contra nuestro schema mySchema usando la herramienta Editix.

En la figura 6.2 se mostrará la jerarquía que tiene nuestro esquema.

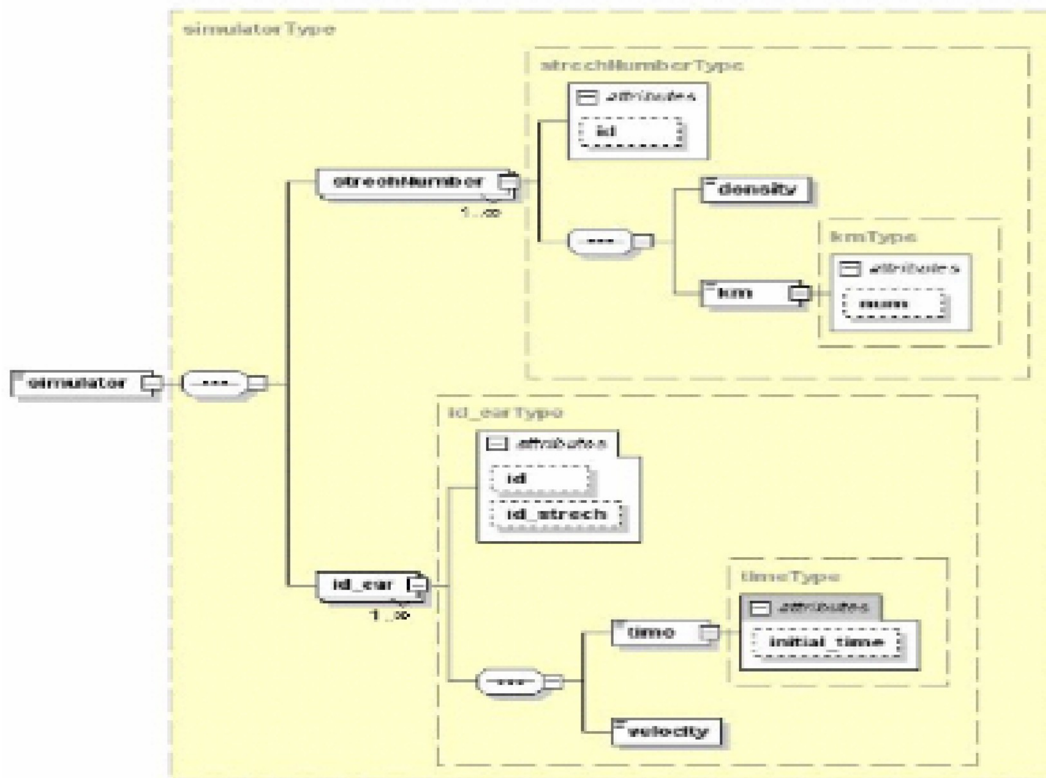

Figura 6.2 Jerarquía del Schema de nuestros documentos GML

En la figura 6.2 se puede observar que tenemos un elemento raíz llamado *Simulator* el cual contiene un tipo complejo en donde se encuentran dos elementos más que son: *id_stretch, id_car* los que también son tipos complejos por contener una secuencia de elementos. En el caso de i*d_stretch* tiene dos elementos de tipo String, y para el caso de *id_car* también tiene 2 elementos más de tipo flota y string. Cada elemento contiene atributos que los identifican como únicos en el documento GML.

A continuación mostraremos un ejemplo de un documento GML que sigue la estandarización de nuestro esquema.

```
  <?xml version="1.0" encoding="iso-8859-1" ?>
- <simulator xmlns="/Users/daxte/Desktop/miscema.xsd" xmlns:po="/Users/daxte/Desktop/miscema.xsd"
orderDate="1999-10-20"> esquema
- <stretchNumber id="1a">
  <density>24</density>
  <km num="598">395,193 379,239 236,233 179,222 143,218 81,238</km>
  </stretchNumber>

- <stretchNumber id="1b">
  <density>24</density>
  <km num="500">406,161 412,123 421,110 429,84 474,34 482,11</km>
  </stretchNumber>

- <stretchNumber id="1c">
  <density>24</density>
  <km num="700">387,166 267,157 229,174 192,185 184,177 167,178 167,196 188,217</km>
  </stretchNumber>

- <stretchNumber id="1d">
  <density>24</density>
  <km num="1000">442,198 482,206 531,262 590,338 604,346 653,410 693,451 727,469</km>
  </stretchNumber>

- <stretchNumber id="1e">
  <density>24</density>
  <km num="800">387,157 393,130 372,63 353,62 342,70 296,21</km>
  </stretchNumber>

- <id_car id="1b" id_stretch="1a">
```

```
<time initial_time="0">5</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1r2" id_stretch="1e">
<time initial_time="0.8">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1s2" id_stretch="1e">
<time initial_time="0.2">8</time>
<velocity>24.5</velocity>
</id_car>

+ <id_car id="1t2" id_stretch="1e">
<time initial_time="0.7">8</time>
<velocity>24.5</velocity>
</id_car>
- <id_car id="1c" id_stretch="1b">
<time initial_time="2.5">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1r" id_stretch="1b">
<time initial_time="0.8">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1s" id_stretch="1b">
<time initial_time="0.2">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1t" id_stretch="1b">
<time initial_time="0.7">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1p" id_stretch="1a">
<time initial_time="1.23">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1q" id_stretch="1a">
<time initial_time="1.5">8</time>
<velocity>24.5</velocity>
</id_car>

- <id_car id="1h" id_stretch="1c">
<time initial_time="0.5">8</time>
<velocity>24.5</velocity>
</id_car>
```

```
- <id_car id="1d" id_stretch="1c">
  <time initial_time="1">12</time>
  <velocity>24.5</velocity>
  </id_car>

- <id_car id="1e" id_stretch="1d">
  <time initial_time="2">15</time>
  <velocity>24.5</velocity>
  </id_car>

  </simulator>
```

Las diferencias entre el documento FIRST y e documento que creamos CENTIA es que en el documento CENTIA alteramos algunos valores para la visualización dinámica, puesto que estos valores no son otorgados por el usuario. Un ejemplo de esto es el tiempo de evacuación y la creación de nuevos carros.

6.3 Transformación de hojas de estilo XSLT a documentos SVG.

Las hojas de estilo se ocuparon en nuestro sistema para poder transformar un documento GML a un documento SVG, la forma en que se realiza esta transformación es mediante una clase implementada en java que se explicara más adelante. En esta sección explicaremos nuestra los módulos de nuestra hoja de estilo.

1. Declaración XML

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

2. Declaración de la hoja de estilo

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

3. Patrón que concuerda con el elemento raíz de nuestro GML, en este caso nuestro nodo raíz es *simulator*

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

4. Sintaxis del documento SVG, declaración de un documento SVG, versión, tamaño del documento (width, heigth), así como namespace referenciados a la estandarización.

```
<svg height="8in" version="1.1" width="10in" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

5. Elemento SVG el cual describe el nombre de una imagen

```
<desc>This graphc links to an external image </desc>
```

6. Se añade una imagen al documento SVG ligada a una liga en la web, se declara el tamaño de la imagen así como su titulo y las coordenadas en donde se coloca dicha imagen.

```
<image height="500px" width="800px" x="135" xlink:href="http://www.udlap.mx/~is113943/map.GIF"
y="46">
<title>Rutas de evacuacion</title>
</image>
```

7. Inicio de un ciclo con una condición (for-each) en donde mencionamos que cada vez que encuentre el elemento *stretchNumber* añadirá al documento una ruta que tendrá como coordenadas la información CDATA del elemento *km,* y tendrá un id que es el atributo del mismo elemento *km*, llamado *id*. Trazando una línea roja de tamaño de grosor 2.06.

```
<xsl:for-each select="stretchNumber">
<path d="{km}" fill="none" id="{@id}" stroke="red" stroke-width="2.06"/>
</xsl:for-each>
```

8. Otro ciclo condición solo que en este caso entrará cada vez que encuentre el nodo *id_car* pondrá en cada ruta de evacuación el tiempo de evacuación de todos los carros y

añadiendo un polígono el cual será la figura de un carro el cual contendrá una animación la cual empezará según el atributo del elemento *time*. La duración de esta animación será dada por los datos CDATA del mismo elemento *time*, no teniendo ninguna repetición ni rotación. Finalizando con una liga hacia la trayectoria o ruta de dicha animación que tiene un nombre según el atributo del elemento *id_stretch*.

```
<xsl:for-each select="id_car">
mensaje del tiempo de evacuación
<text class="Label" style="font-size:10">
<textPath dy="10" startOffset="25%" xlink:href="#{@id_stretch}">
<xsl:value-of select="time"/>
</textPath>
</text>
<g id="{@id}" opacity=".7">
figura de carro
<polygon fill="#FAA60E" onmousemove="DoOnMouseMove(evt)"
onmouseout="circle_click2(evt)" onmouseover="circle_click(evt)" points="0,9.187 0,0.596 2.193,0.596
2.193,4.035 3.511,4.035 3.511,2.525 5.919,2.525 8.551,3.716      12.274,3.702 14.911,4.892 14.911,0.596
16.008,0.596 17.102,1.46 17.102,8.33 16.008,9.187 14.911,9.187 14.911,4.892      12.274,6.082 8.551,6.098
5.919,7.286 3.511,7.286 3.511,5.775 2.193,5.775 2.193,9.187 "/>
<path d="M9.429,4.876c0-0.519-0.28-0.941-0.621-0.941c-0.351,0-0.631,0.423-
0.631,0.941c0,0.527,0.28,0.951,0.631,0.951     C9.148,5.827,9.429,5.403,9.429,4.876z"/>
<rect fill="#404040" height="2.36" width="3.995" x="2.635"/>
<rect fill="#404040" height="1.859" width="3.498" x="9.429" y="1.314"/>
<rect fill="#404040" height="2.36" width="3.995" x="2.635" y="7.46"/>
<rect fill="#404040" height="1.859" width="3.498" x="9.496" y="6.651"/>
anmacion del carro
<animateMotion begin="{time/@initial_time}" dur="{time}" repeatCount="0" rotate="auto">
<mpath xlink:href="#{@id_stretch}"/>
</animateMotion>
</g>
</xsl:for-each>
```

9. Cerramos el documento SVG

```
</svg>
```

10. Cerramos el patrón *templete* y la hoja de estilo

```
</xsl:template>
```

```
</xsl:stylesheet>
```

Nota 1: para poder encontrar a los atributos de los elementos anteponemos el símbolo '@', así como para hacer consultas se tiene que encerrar entre corchetes '{ }'.

Nota 2: Para darle una mejor visión a nuestro documento SVG añadimos un script en donde cambiamos de color los vehículos si pasan el Mouse sobre el, así como también las rutas. La hoja de estilo, la salida del documento SVG y el script se encuentran en el anexo.

6.4 Paquetes Java

6.4.1 Modelo:

En este módulo encontramos 3 archivos los cuales se dividen en un archivo para recorrer y almacenar el documento GML, otro archivo para crear un documento GML, y por ultimo un archivo para transformar y crear un documento SVG.

Paquete DOMgml:

Esta clase tiene como función primordial es la del parseo del documento GML que se no propicio por el instituto FIRST.

A continuación se muestra una parte del código en donde parseamos el documento gml:

En este método accedamos al documento GML y lo cargamos a la clase *DOMgml* y le mandamos en forma de nodo a nuestro método *traverse_tree:*

```
public void computeGrades(String uri)  {
try  {
                DOMgml parser = new DOMgml();          inicializa parser
                parser.parse(uri);                      se le manda el documento a parsear
                traverse_tree(parser.getDocument());   Mandamos a llamar al método y le pasamos el nodo del parser
        } 
catch (Exception e) 
{
 e.printStackTrace(System.err);
}
  }
```

Después de recibir el nodo tendremos que verificar si es un elemento o si es texto. A continuación se muestra el código donde verificamos el tipo de nodo, y dependiendo el tipo de nodo es la información que buscamos, es decir que si es un nodo de tipo elemento buscaremos elementos de nuestro documento GML como lo puede ser *stretchNumber*:

```
int type = node.getNodeType();
switch (type)
{
        case Node.ELEMENT_NODE:                 SI ES EL NODO DE TIPO ELEMENTO
                {
                        String elementName = node.getNodeName();
                        gi = -1;
                        NamedNodeMap attrs = node.getAttributes();
                        if(elementName.equals("stretchNumber"))   SI EL ELEMENTO TIENE COMO NOMBRE stretchNumber
                        {
                                Attr attrib = (Attr)attrs.getNamedItem("id");   DAME EL ATRIBUTO ID
                                id_stretch[cont1][0] = attrib.getValue();     ALMACENO EL ATRIBUTO EN UN ARREGLO
                                cont1++;
                        }
      case Node.TEXT_NODE:
                {………….}
```

En este método verificamos todos los elementos y atributos de este mismo y los guardamos en arreglos los cuales mandaremos a la siguiente clase para que pueda procesarlos, el método que realiza esta llamada es *CreaGML* o *CreaGMLDinamico*.

La diferencia entre estos dos métodos es que uno crea una visualización estática y el otro una visualización dinámica.

```
CreaGML documento=new CreaGML();
                    documento.creaDocumento(id_stretch,velocity,km,contador,-1,-1);    VISUALIZACION ESTATICA
CreaGML documento=new CreaGML();
          documento.creaDocumento(idstretch,velocity2,km,contador,tiempoinicial,tiempofinal); VISUALIZACION DINAMICA
```

Paquete CrearGML

El objetivo principal de esta clase es el de crear otro documento GML a partir del ofrecido por el instituto FIRST, modificando alguna información.

El método más importante de esta clase es el de *crearGML* el cual recibe toda la información que se parseo de la clase *DOMgml*. Este método crea elementos, añade atributos a este mismo y coloca texto en el mismo elemento. A continuación se muestra el código de este método:

```
Element root = new Element("simulator");
                    String tiempo;
                    for (int i = 0; i < cont[0]; i++) {
                              Element stretchNumber = new Element("stretchNumber"); CREA UN ELEMENTO
                              stretchNumber.setAttribute("id", idst[i][0]);   AÑADE UN ATRIBUTO AL ELEMENTO
                              Element density = new Element("density").setText(idst[i][1]);
                              stretchNumber.addContent(density);
                              Element km = new Element("km");
                              km.setText("M"+kmt[i]);  INSERTA VALORES EN EL CAMPO CDATA DEL ELEMENTO
                              km.setAttribute("num",Integer.toString(i));
                              stretchNumber.addContent(km);
                              root.addContent(stretchNumber);
                              tiempo=regresaTiempo(kmt[i],velocityt[i]);
```

En este método lo que se realiza por medio de librerías JDOM es la creación de elementos, que son nodos JDOM, en los cuales se les va añadiendo propiedades como atributos texto, dejando al final la incorporación de hijos (nodos subyacentes).

E esta clase no solo creamos un documento si no también modificamos información como es el caso del cálculo del tiempo de recorrido de todos los vehículos que transitan por una ruta. Este cálculo se hace mediante el método *regresaTiempo* que recibe la velocidad de los vehiculo así como la distancia de las rutas de evacuación Con esta información calcularemos el tiempo de tardado, mediante la formula siguiente:

$$t=d/v$$

A continuación se muestra el código de esta operación:

```
public String regresaTiempo(String coord, String velocidad){
                int distancia=Integer.parseInt(coord);
                float tiempo=(distancia/Float.parseFloat(velocidad));  FORMULA
                if(tiempo<0)
                        tiempo=-(tiempo);
                return Float.toString(tiempo);    REGRESAMOS RESULTADO
        }
```

Este método es llamado cada vez que se añade una ruta al nuevo documento GML. Con esto obtenemos un tiempo de recorrido por cada ruta, queriendo decir que todos los vehículos viajaran a una misma velocidad en cada ruta.

Por último tenemos el punto más importante que es la creación en el disco duro del nuevo documento GML esta operación la hace otro método de esta misma clase, el cual recibe un nodo que es el nodo que contiene la información del documento del instituto FIRST y las modificaciones realizada por la clase pasada:

```
public  void guardaDocumento(Element root){ RECIBE EL NODO RAIZ QUE CONTIENE TODOS LOS ELEMENTOS
                Document doc = new Document(root);//Creamos el documento
                //Vamos a almacenarlo en un fichero y además lo sacaremos por pantalla
                try {
                        XMLOutputter out = new XMLOutputter("  ", true);
```

```
                    FileOutputStream file = new FileOutputStream("/Library/Tomcat/webapps/tesis/xml/CENTIA.xml");
                    MANDAMOS LA RUTA DE DONDE QUEREMOS GUARDAR EL NUEVO ARCHIVO
                    out.output(doc, file);
                    file.flush();
                    file.close();
                    out.output(doc, System.out);
            } catch (Exception e) {
                    e.printStackTrace();
            }
```

Paquete SimpleTransform

Esta clase es sencilla puesto que solo tiene un método el cual accede a disco y cargará un documento GML y una hoja de estilo:

```
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer(new StreamSource("/Library/Tomcat/webapps/tesis/WEB-INF/classes//myxsl.xsl"));

                    MANDAMOS LA HOJA DE ESTILO PARA LA TRANSFORMACION

transformer.transform(new StreamSource("/Library/Tomcat/webapps/tesis/WEB-INF/classes/xml/ejemplo.xml"),new StreamResult(os));

                    MANDAMOS EL DOCUMENTO XML PARA LA TRANFORMACION
```

Después de la transformación tendremos que guardarlo en un archivo SVG para poder accederlo desde una pagina web:

```
File newXML = new File("/Library/Tomcat/webapps/tesis/servlets/mysvg.svg");

                    CREAMOS UN NUEVO DOCUMENTO SVG

FileOutputStream os = new FileOutputStream(newXML);
StreamResult result = new StreamResult(os);
```

Finalmente, el módulo del modelo, es el encargado de parsear documentos, modificar esa misma información y añadirla en otro nuevo documento, que después será transformado por una hoja de estilo y creará un nuevo documento SVG, todo esto en 3 clases.

6.4.2 Controlador

En este módulo ocuparemos 3 archivos 2 que serán servlets y una clase, los servlets serán encargados de hacer el llamado al módulo vista, y la clase será la encargada de llamar al módulo modelo.

Paquete Controlador

Esta clase solo contiene 2 métodos, el primero realiza una visualización estática, el cual manda a llamar a métodos que solo realizan la tarea de parsear el documento del instituto FIRST y no ingresa información adicional. Para poder ejecutar esta acción, este método tiene que llamar a dos clases del módulo modelo que son *DOMgml y SimpleTransform*

La razón de la construcción de una clase que actué como controlador y no hacerlo directamente con el servlet, se debió a que así podremos ejecutar nuestro software de una manera standalone y vía Web, es decir que podríamos correr nuestro programa sin conexión a la red o con conexión.

```
public  void visuaEstatico(){
        DOMgml simulator=new DOMgml();
simulator.computeGrades("/Library/Tomcat/webapps/tesis/xml/FIRST.xml"); MANDAMOS DOCUMENTO A PARSEAR
        simulator.crearGML();
        try{
                SimpleTransform w=new SimpleTransform();  TRANSFORMAMOS A SVG
                w.transformacion();
        }   catch (Exception e) {
                e.printStackTrace(System.err);
        }

}
```

El siguiente método es para la creación de una visualización dinámica, para poder realizar esta tipo de visualización se necesita de información como lo son los parámetros: cantidad de vehículos, su velocidad, tiempo inicial, tiempo final. La diferencia de este método con el estático son estos parámetros, por esta razón mandamos esa información a otro método de la clase *DOMgml*

```
public  void visuaDinamico(String cantidad[][],String velocidad[],int tiempoinicial,int tiempofinal){
			DOMgml simulator=new DOMgml();
			simulator.computeGrades("/Library/Tomcat/webapps/tesis/xml/FIRST.xml"); MANDAMOS DOC A PARSEAR
			simulator.crearGMLdinamico(cantidad,velocidad,tiempoinicial,tiempofinal);   CREAMOS DOCUMENTO
			try{
				SimpleTransform w=new SimpleTransform(); TRANSFORMAMOS A SVG
				w.transformacion();
			}   catch (Exception e) {
				e.printStackTrace(System.err);
			}

		}
```

Paquete Estático

Este servlet se encarga de mandar a petición del usuario. Si el usuario quiere una visualización estática mandará a llamar a la clase *controlado*r, pero si requiere una visualización dinámica tendrá que llamar al JSP del módulo vista, puesto que para hacer una visualización dinámica es necesario llenar una forma html además de llamar a la clase *controlador* . El JSP necesitará de información del módulo de control.

```
if(paramName.equals("estatico"))   SI EL USUARIO ELIGE VISUALIZACION ESTATICA
			{
				Controlador control=new Controlador();
				control.visuaEstatico(); MANDAMOS A LLAMAR A LA VISUALIZACION ESTATICA
				RequestDispatcher   view   =   request.getRequestDispatcher("/jsp/vistaEstatica.jsp");   QUIEN   RECIBIRA
REQUEST
				view.forward(request, response); MANDAMOS REQUEST Y RESPONSE
			}
			if(paramName.equals("dinamico"))
			{
```

```
Controlador control=new Controlador();
String temp=Integer.toString(control.regresaNumeroRutas()); VISUALIZACION DINAMICA
Object result=(Object)temp;
request.setAttribute("#ruta",result); AÑADIMOS ATRIBUTO Y LE PONEMOS DE VALOR EL
NUMERO DE RUTAS DE EVACUCION
RequestDispatcher view = request.getRequestDispatcher("/jsp/vistaDinamica.jsp");DECIMOS QUIEN
RECIBIRA EL REQUEST
view.forward(request, response);
```

Para poder mandarle información al JSP es necesario añadir información a la variable request del servlet, el único imprevisto es que tiene que ser un objeto y por tal razón se hizo un casting a String y luego a objeto.

```
Object result=(Object)temp;
request.setAttribute("#ruta",result);
RequestDispatcher view = request.getRequestDispatcher("/jsp/vistaDinamica.jsp");
view.forward(request, response);
```

Paquete Dinámico

Su función es el de abstraer la información de una página para poder construir una visualización dinámica. Esta información será utilizada por la clase *controlador* que se encargara de mandar a llamar al módulo modelo. Por esta razón la debemos de extraer y mandársela, para después mandar la respuesta al siguiente módulo que es la vista.

A continuación se muestra como abstraemos la información mediante condiciones:

```
if(paramName.equals("cantidad"+Integer.toString(i))){
        if(paramValue.equals(null) || paramValue.equals(""))
                band++;
        else
        {
        id_stretch[i][1]=paramValue;
        id_stretch[i][0]=Integer.toString(i);
```

```
				}
			}
			if(paramName.equals("velocidad"+Integer.toString(i))){
				if(paramValue.equals(null) || paramValue.equals(""))
					band++;
				else
					velocidad[i]=paramValue;
			}

			if(paramName.equals("tiempoinicial")){
				if(paramValue.equals(null) || paramValue.equals(""))
					tiempoinicial=-1;
				else
					tiempoinicial=Integer.parseInt(paramValue);
			}
```

Para después pasar esta misma información al módulo de control mediante la clase controlador que se encargará de llamar a las clases correspondientes:

```
control.dinamico(id_stretch,velocidad,tiempoinicial,tiempofinal);
```

Después de llamar al la clase controlador mandaremos la repuesta al módulo de las vistas:

```
RequestDispatcher view = request.getRequestDispatcher("/jsp/vistaEstatica.jsp");
			view.forward(request, response);
```

6.4.3 Vista

La función principal de este módulo es el de mostrarle al cliente los resultados o la forma para enviar una petición, la maneja de poder mostrarle esta información es mediante JSP's. Para nuestra aplicación creamos 3 JSP's los explicaremos a continuación.

VistaEstatica

En este JSP lo que mostramos es un documento HTML que accede a disco, buscando un documento SVG, mostrándolo en una pagina web.

```
<html>
	<BODY text=#000000 bgColor=#9999CC scroll=no>
	<font color='yellow'>warning:</font><font color='#FFFFFF'> if you dont view this image, download the SVG viewer <a
			href='http://www.adobe.com/svg/viewer/install/main.html'>http://www.adobe.com/svg/viewer/install/main.html</
			a></font>
		<embed name='svg0' type='image/svg+xml' width='800' height='500' src='http://localhost:8080/tesis/servlets/mysvg.svg'>
			</embed>
		<noembed>Appropriate text equivalent.</noembed>
	</body>
</html>
```

La imagen es la siguiente figura 6.3:

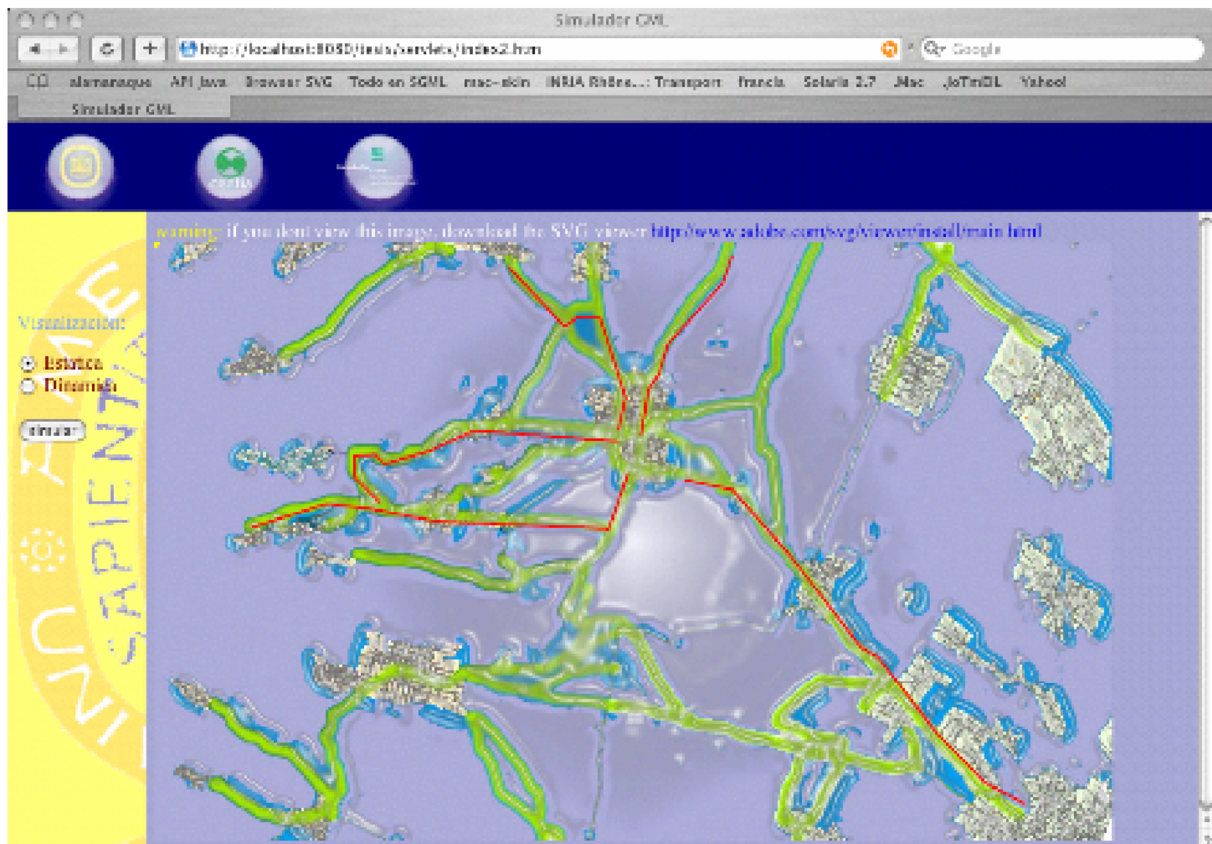

Figura 6.3 interfaz de la visualización Estática de nuestro sistema

VistaDinamica

Este JSP es mucho más complicado que el pasado, puesto que en este JSP utilizamos información que nos fue enviada del servlet. Dicha información es el número de rutas que contiene nuestro documento GML, que nos ayudará a construir una forma HTML. Esto es necesario para que el usuario pueda llenar los campos de la forma y mandar otra petición que será la petición de a visualización dinámica.

```
<html>
        <head>
        <title>Simulacion Dinamica</title>
        <BODY text=#000000 bgColor=#9999CC scroll=no>
        <center>Pagina JSP</center>
        <form action='/tesis/Dinamico' method='post'>
<%
        String temp= (String)request.getAttribute("#ruta");
        int cant=Integer.parseInt(temp);
        for(int i=0;i< cant;i++){
            out.println("<font color='#FFFFFF'>"+
            "<label>numero de carros en la ruta "+i+" <input name='cantidad"+i+"' type='text' size='5'
                maxlength='5'>\n"+
            "<label>velocidad de la ruta "+i+"</label>\n"+
            "<input name='velocidad"+i+"' type='text' size='3' maxlength='3'> <br>\n");
        }
%>


        <p>
        <label>Ingresar más informacion
        <br>
        <input name='otros' type='checkbox' value='otros'>
        </label>
        </p>
        <p>
        <label>Ver solo vehiculos cuyas salidas sea en los segundos: </label> <br>
        <input name='tiempoinicial' type='text' value='-1'size='3' maxlength='3'>
        <label>entre</label></font>
        <input name='tiempofinal' type='text' value='-1' size='3' maxlength='3'>
        <br>
        <br>
        <input type='submit' name='Submit' value='Simular'>
        </p>
        </form>
        </body>
</html>
```

La imagen es la siguiente figura 6.4:

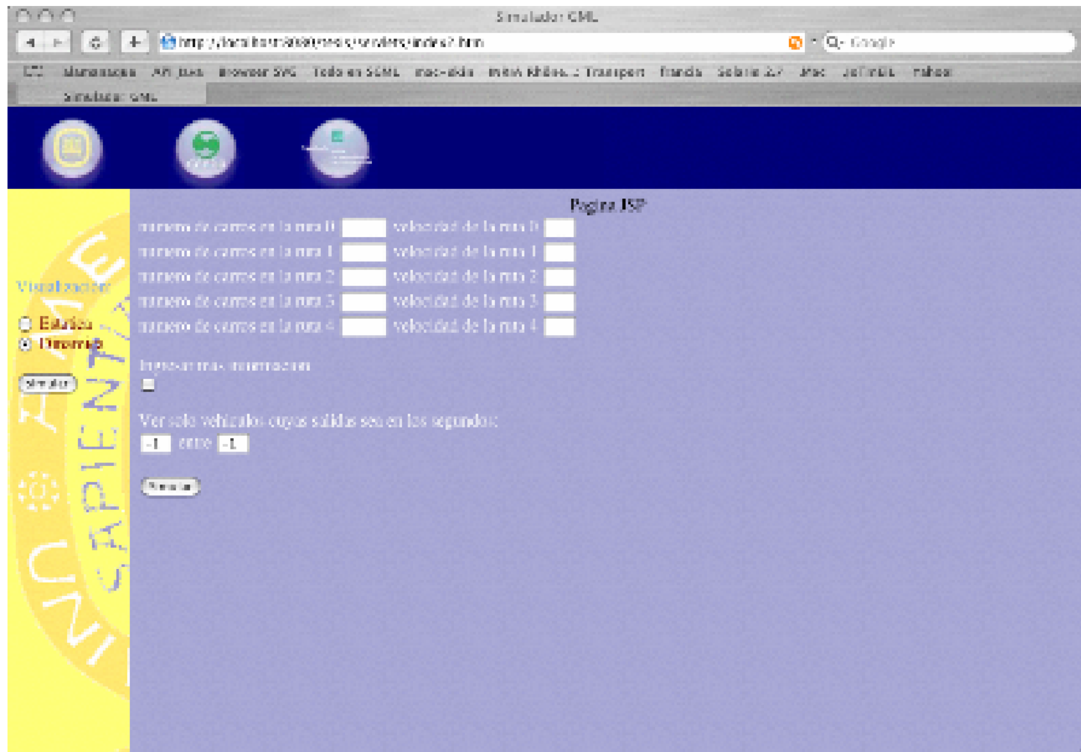

Figura 6.4 interfaz de la visualización Dinámica de nuestro sistema

vistaError

Este JSP lo que nos muestra es un mensaje de error de valor nulo, es decir si la forma de la vistaDinamica no es llenada correctamente imprime un mensaje de advertencia. La forma de no llenar correctamente la forma es dejar un campo de un *texfield* vació o con valor nulo.

```
<html>
        <BODY text=#000000 bgColor=#9999CC scroll=no>
                <font color='yellow'>Advertencia:</font> <font color='#FFFFFF'>error no ingreso algun
                        valor</font>
                <form action='/tesis/jsp/vistaDinamica.jsp' method='post'>
                <input type='submit' name='regresar' value='regresar'>
                </p>
                </form>
```

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

La imagen es la siguiente figura 6.5:

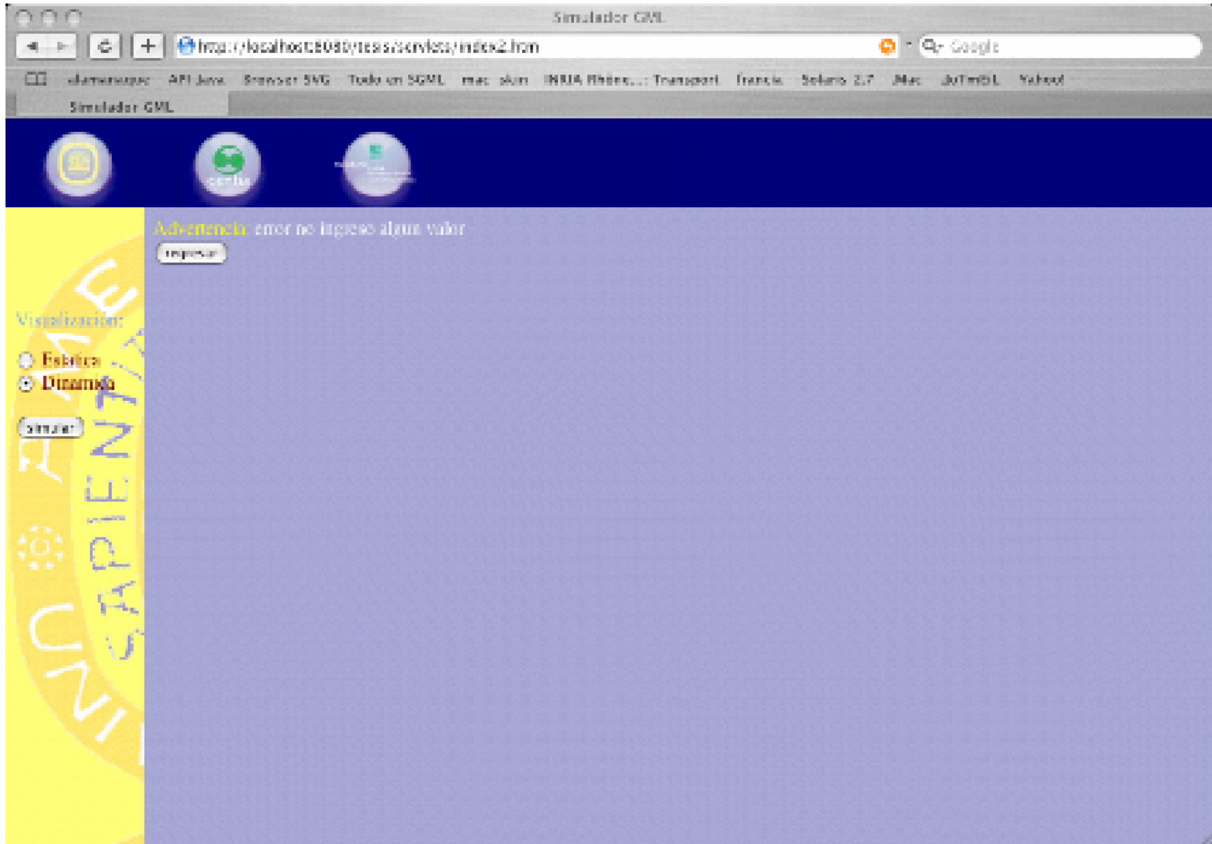

Figura 6.5 interfaz de l mensaje de error de nuestro sistema

6.5 Problemas con la Implementación

Los problemas que salieron al implementar nuestro software es el de un plug in para poder visualizar documentos SVG, puesto que ningún navegador lo tiene instalado, el problema fue que para poder visualizar nuestra aplicación se necesitará instalar primero el plug in.

Otro problema que se presento fue el de los navegadores puesto que ocupamos navegadores como el Safari, netscape o opera los cuales al hacer la visualización con el plug in se tardan mucho en ejecutar un zoom in o en cargar el documento SVG.