GML y JDOM

CAPÍTULO 4

CAPÍTULO 4 “GML y JDOM”

El intercambio de información siempre ha sido un problema cuando se utilizan lenguajes y sistemas operativos heterogéneos. Este problema se vio más claramente con la llegada de Internet, de ahí la importancia de un estándar abierto para representar información como lo es GML. GML como tal no es suficiente para resolver esto, también necesitamos maneras de procesarlo y es ahí donde herramientas como JDOM entran en acción. En este capítulo se explicarán los conceptos de GML y JDOM, se describirán la historia, definiciones y predecesores de cada uno.

4.1 GML

Geography Markup Language (GML) es una codificación estándar basada en XML (*Extensible Markup Language*) [W3, 2004] para el modelado, transporte y almacenamiento de información geográfica (posición, localización, extensión, etc.) incluyendo propiedades espaciales y no espaciales de las *geographic features* (Entidades geográficas). [OpenGIS GML, 2002] Fue desarrollado por el *OpenGIS Consortium* (OGC) y uno de sus objetivos es permitir a los navegadores la habilidad para ver *web based mapping* (Cartografía en la *web*) sin componentes adicionales o visualizadores. Está considerado como un *open data exchange standard*, (Estándar para el intercambio de información abierta) muy adecuado para transmitir diferentes volúmenes de información. En la figura 4.1 podemos ver un pequeño ejemplo de un documento en GMLv2.1.2. Este ejemplo fue generado por el software GISonline [Cepeda, 2003]. Los *namespaces* son los de gisonline y gml, además se trata de un ejemplo referente a un tramo.

Figura 4.1 Documento en GML [Cepeda, 2003]

4.1.1 Historia

La versión 1.0 de GML fue publicada en Mayo del 2000 y ofrecía tres diferentes perfiles llamados GML 1, GML 2 y GML 3. Dichos perfiles eran construcciones débiles sobre GML 1.0 ya que traslapaban diferentes métodos para codificar (XML 1.0 (DTD) y RDF) con diferentes formas de codificar los esquemas.

Para Febrero 20 del 2001 el *OpenGIS Consortium* publicó la versión 2.0, dando las bases para el desarrollo de una *WWW* geoespacial. La versión 2.0 está basada completamente en *XML Schema*. Fue para el 17 de septiembre del 2002 que la especificación de GML versión 2.1.2[1] fue publicada por *OpenGIS*.

4.1.2 Versiones de GML

La versión 2.0 está basada completamente en el *XML Schema*[2], a diferencia de la 1.0 que estaba basada en una combinación de XML DTD's (*Document Type Definition*) y *Resource Description Framework* (RDF). GML 2.0 provee un único método de

[1] Versión que está siendo utilizada en esta tesis.
[2] Definición del formato de un documento GML. Soporta type inheritance, distributed schema integration y namespaces

codificación (*XML Schema*) y un método único para la creación de *feature schemas*[3]. En la versión 2, sólo se manejan "simple features", aquellas features que están restringidas a geometrías en 2D, donde la delimitación de una curva está sujeta a interpolación linear. Dentro de las diferencias entre versiones, encontramos que GML 1.0 no aceptaba los *namespaces*, el *feature type* no estaba definido y no ofrecía medios para la representación de un *feature schema* independiente de la instancia de los datos, mientras que GML 2.0 sí. Estas diferencias se pueden ver en las figuras 4.2 y 4.3.

En el ejemplo de la versión 1 (figura 4.2) se puede ver que no se usan *namespaces*. No se pueden definir tipos propios, sino que se debe recurrir al *typeName*. Las geometrías se deben definir como *geometricProperty*.

En el ejemplo de la versión 2 (figura 4.3) se puede ver los *namespaces* pue y gml. Aquí se pueden definir tipos propios como son *Road* y *numberLanes*. Las geometrías se definen con *gml:centerLineOf* en lugar de con *geometricProperty* como en la versión 1.

En la versión de GML 3.0 se hicieron las siguientes mejoras:

- Representar fenómenos geoespaciales, incluyendo *features* con geometrías no lineales en 3D, *features* con topologías[4] en 2D, *features* con propiedades temporales, *features* dinámicas, coberturas y observaciones.
- Proporcionar más apoyo para las propiedades de los *features* y otros objetos cuyo valor es complejo.
- Usar un sistema de referencia, unidades de información estándar en la representación de un fenómeno geoespacial, observaciones y valores.

[3] Esquema para definir entidades (*features*).
[4] Topología: La relación entre *features*.

- Representar en el espacio con sistemas temporales de referencia, unidades de medición y estándares de información.
- Representar estilos predefinidos para un *feature* y su visualización.
- Acoplarse a otros estándares[5] como [ISO/TC 211, 2004]:
    - ISO DIS 19107 *Geographic Information – Spatial Schema*
    - ISO DIS 19108 *Geographic Information – Temporal Schema*
    - ISO DIS 19118 *Geographic Information – Encoding*
    - ISO DIS 19123 *Geographic Information.*

En el ejemplo de la figura 4.4 se pueden ver nuevos elementos como *Segment*, *CubicSpline*, id para la geometría Curve y el uso de vectores.

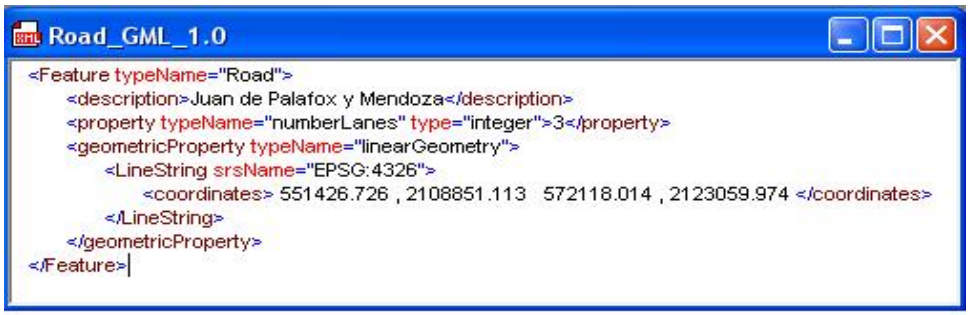

Figura 4.2 Ejemplo de la representación de un camino en GML 1.0 [Razo, 2001]

Figura 4.3 Ejemplo de la representación de un camino en GML 2.0 [OpenGis GML, 2002]

[5] DIS = *Draft International Standards*

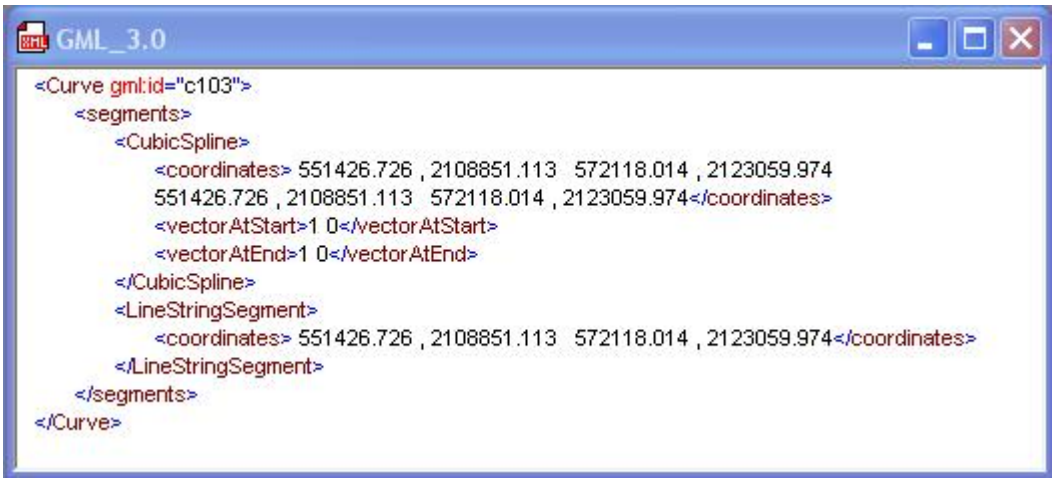

Figura 4.4 Ejemplo de una parte de un documento en GML 3.0

4.2 GML y XML

Debido a que GML está basado en XML, éste hereda todas las ventajas por las cuales XML es tan popular. GML puede ser manipulado con todas las herramientas estándares de XML. Ambos son llamados lenguajes de marcado "*markup*" o meta-lenguajes.

GML, al igual que XML, separa el contenido de la presentación, enfocándose sólo en la captura del contenido geográfico y logrando así que ésta sea una herramienta para la descripción de datos geográficos y la administración de documentos que contienen información geográfica. Ya que GML sólo se preocupa por la descripción del contenido geográfico y sus atributos no espaciales, éste debe ser estilizado para su presentación. La presentación puede implicar que sea estilizado en forma gráfica, como un mapa o como texto e incluso como una secuencia de instrucciones de voz.

XML proporciona un mecanismo para enlazar múltiples recursos dentro de asociación compleja. Los enlaces (*links*) de XML pueden ser recorridos en ambas

direcciones y permiten asociar elementos XML o incluso fragmentos de elementos. Estas características de XML ayudan a que GML tenga la habilidad para construir asociaciones entre *spatial features* (Entidad espacial). Con todos los tipos de datos que se representan en XML, uno de los objetivos es que con GML se puedan asociar datos geoespaciales con sus atributos no espaciales (texto, video, imágenes, etc.).

GML proporciona una variedad de tipos de objetos para describir información geográfica incluyendo *features*, sistemas de referencia de coordenadas, geometrías, topologías, tiempo, unidades de medición y valores generalizados.

4.3 Ventajas de GML

De acuerdo a la compañía *Galdos Inc*. [Galdos, 2004] las principales ventajas por las cuales debe usarse GML son las siguientes:

1.- Funciona en un navegador sin necesidad de otro software. Cuando un archivo GML es recibido del lado del cliente, éste es convertido en un conjunto de dibujos de objetos y los despliega como un mapa en el navegador. Comúnmente se usa *Scalable Vector Graphics* (SVG) como el lenguaje para dibujar los features.

2.- Estilo personal de la presentación. Ya que GML tiene sólo contenido, se le pueden aplicar diferentes *stylesheets*[6] para poderlo ver como se desee.

3.- Mapas editables. Una vez que GML ha sido convertido en SVG, el usuario puede utilizar herramientas de edición de gráficos del lado del cliente para agregar texto, resaltar *features* o cualquier otra cosa.

[6] Hojas de estilo.

4.- Sofisticadas capacidades para hacer links. Se puede asociar cualquier página web con un *feature*.

5.- Mayor capacidad para realizar consultas. Comúnmente los usuarios quieren darle clic a un *feature* y saber más de él, para un mapa en formato GIF esto sería una tarea muy difícil pero con GML se vuelve muy sencilla.

6.- Control sobre el contenido. Debido a que GML es *feature-based*, es sencillo proporcionar un filtro que permita al usuario descargar sólo las *feature-types* que deseen que aparezcan en su mapa.

7.- *Features* animados. Objetos y features que cambian con el transcurso del tiempo pueden ser agregados en GML.

8.- No se tiene que utilizar sólo un *browser*. Por ser un formato abierto, GML se puede manipular en PDA's y celulares por dar un ejemplo.

9.- Encadenamiento de servicios. Un ejemplo de encadenamiento de servicios es cuando se toma información geoespacial y se manda a un sitio para ser convertida, por ejemplo, del sistema de referencia NAD27, al sistema de referencia NAD83, de ahí se manda a otro sitio para convertir las coordenadas de coordenadas geográficas a UTM, de ahí se manda a un sitio distinto para agregarle información demográfica y finalmente se manda a otro sitio para ser desplegada o almacenada.

4.4 Elementos de un Documento GML

Todo documento de GML debe o puede contener (según sea el caso) los siguientes elementos, la representación en GML se puede ver en las imágenes del apéndice C en la sección 1.

- La versión de XML que se está utilizando.
- Tipo de Codificación de XML. Esta codificación puede ser US-ASCII (7 bits), UTF-8 (código Unicote), UTF-7 o ISO-8859-1.
- Definición de los *namespaces*. En este caso podemos ver que están definidos los namespaces de gml, gisonline y xsi.
- Un solo elemento raíz. Forzosamente debe existir sólo un elemento inicial que englobe a todos los demás.
- *Bounding Box*. Contiene las coordenadas Xmin, Ymin, Xmax y Ymax, las cuales representan los límites de la capa que se está representando en ese documento.
- Información no geométrica. Atributos o datos descriptivos.
- Información geométrica. Geometrías como punto, polilínea, etc.

4.5 Las geometrías y su codificación en GML

Las geometrías están compuestas por tres grupos de figuras básicas tales como punto, línea, polígono. Su localización está determinada por coordenadas que pueden estar representadas como un elemento <*coord*> o un elemento <*coordinates*>. A continuación representa cada una de las geometrías, así como los elementos de coordenadas, en GML.

De acuerdo al modelo de *simple features* del OGC, GML proporciona elementos geométricos correspondientes a las clases geométricas *point, lineString, linearRing, polygon, multiPoint, multiLineString, multiPolygon, multiGeometry*. Estas clases están descritas a continuación y hacen referencia a las clases descritas en la sección 2.3.3.

- <*coord*> Se representa mediante dos elementos numéricos llamados <*x*> y <*y*>

Figura 4.5 Elemento *coord* [OpenGIS GML, 2002]

- <*coordinates*> Se representa como una cadena de números en donde las coordenadas x y y se separan por una coma y un espacio separa una coordenada de otra.

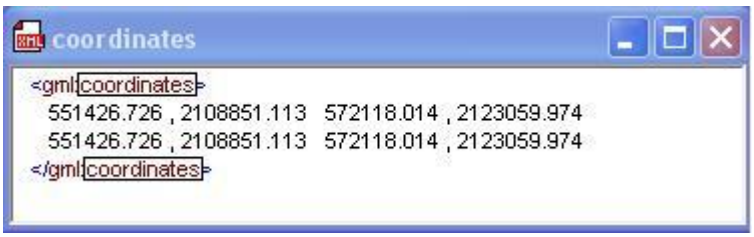

Figura 4.6 Elemento *coordinates* [OpenGIS GML, 2002]

A partir de esta geometría todas las coordenadas se representaran como un elemento *coord*.

- <*point*> Se usa para codificar un punto con un solo par de coordenadas (x, y) en el plano cartesiano.

Figura 4.7 Elemento *point* [OpenGIS GML, 2002]

- <*LineString*> Se usa para codificar una secuencia de líneas rectas, un camino cerrado se indica cuando la última coordenada (x, y) es igual a la primera.

Figura 4.8 Elemento *lineString* [OpenGIS GML, 2002]

- <*MultiLineString*> Se usa para codificar una colección de *lineStrings*. Su elemento de pertenencia es un *lineStringMember*.

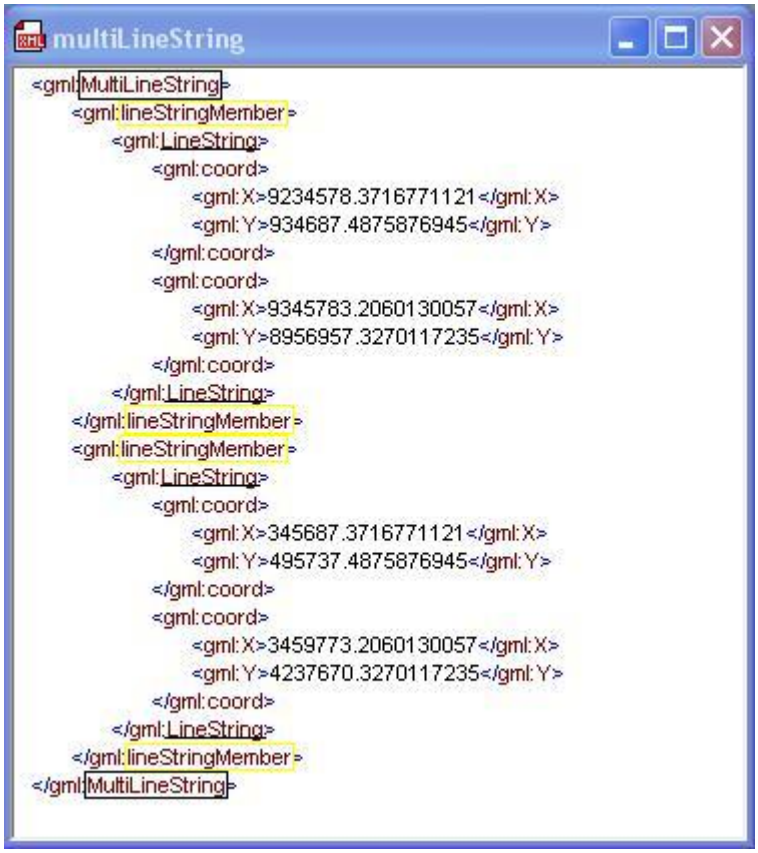

Figura 4.9 Elemento *multiLineString* [OpenGIS GML, 2002]

A partir de estas geometrías sólo se va a dar la descripción textual ya que no son necesarios más detalles debido a que no fueron utilizadas en el presente proyecto. Si se desea conocer la representación en GML se puede consultar el apéndice C en la sección 2.

- <*LinearRing*> se usa para codificar una secuencia cerrada de líneas rectas. Deben coincidir la primera y la última coordenada (x, y).

- <*Polygon*> Se usa para codificar polígonos compuestos por un conjunto de *linearRings*. Un polígono debe tener un límite exterior <*outerBoundaryIs*> y cero o varios límites interiores <*innerBoundaryIs*>.

- <*MultiPoint*> Se usa para codificar un conjunto de puntos. Su elemento de pertenencia es un *pointMember*.

- <*MultiPolygon*> Se usa para codificar una colección de polígonos. Su elemento de pertenencia es un *polygonMember*.

- <*MultiGeometry*> Se usa para codificar un conjunto de geometrías de diferentes tipos. Su elemento de pertenencia es *geometryMember*.

4.6 JDOM

JDOM es un API (*Aplication Programming Interface*) independiente del *parser*, basado en una estructura de árbol, para procesar documentos XML con Java que dejó a un lado las limitaciones de DOM[7] (*Document Object Model*) y empezó de cero. Se trata de un enfoque abierto para parsear, crear, manipular y serializar documentos XML. Es una herramienta en Java, única en su clase, para trabajar con XML y creada para permitir el rápido desarrollo de aplicaciones de XML, además de que es mucho más "limpio" y sencillo que DOM. [Evans, 2001]

[7] Para mayor información, referirse al trabajo de Gerardo Cepeda sección 4.1.2 [Cepeda, 2003]

JDOM usa las convenciones de codificación y librerías de clases. Por ejemplo, todas las clases principales de JDOM tienen los métodos de *equals( ), toString( )* y *hashCode( )*. Todas implementan las interfaces *Cloneable* y *Serializable*. Los hijos de un elemento o documento se guardan en una lista (*java.util.List*). [Harold, 2002]

4.6.1 Historia

JDOM fue inventado por Brett McLaughlin y Jason Hunter, junto con la colaboración de James Duncan Davidson, bajo la licencia de *Apache-like*, en la primavera del 2000. Lo diseñaron estrictamente para XML, totalmente en java y sin ninguna preocupación por tener compatibilidad con API's similares.

4.6.2 Características

Al igual que su predecesor DOM, JDOM representa el documento XML como un árbol compuesto de elementos, atributos, comentarios, instrucciones de proceso, nodos de texto, secciones de *CDATA*, etc. El árbol completo está disponible en todo momento y se puede accesar cualquier parte en cualquier momento. A diferencia de DOM, en JDOM todos los tipos de nodos están representados por clases concretas en lugar de interfaces.

JDOM no incluye por si mismo un *parser*. En vez de eso depende de un *parser* de SAX con un manejador de contenido común para parsear documentos y construir modelos JDOM a partir de estos. JDOM viene con *Xerces 1.4.4*, pero puede trabajar de igual manera con cualquier *parser* compatible con *SAX2* incluyendo *Crimson, AElfred*, el *parser* de Oracle XML para Java, *Piccolo, Xerces-2*. Cualquiera de éstos puede leer un documento XML y ponerlo en JDOM y puede también convertir objetos documento de DOM en objetos documento de JDOM. Es también útil para hacer un pipe de la salida de programas DOM existentes en la entrada de un programa en JDOM. Sin embargo, si se está trabajando con un flujo de datos de XML que se lee de un disco o de la red, es

preferible usar SAX para producir el árbol JDOM y así evitar la sobrecarga de construir el árbol en memoria dos veces en dos representaciones diferentes. [Rusty, 2002]

Una vez que el documento está cargado en memoria, ya sea porque fue creado de raíz o parseado de un *stream* (Flujo de información), JDOM puede modificar el documento. Un árbol de JDOM es de lectura y escritura, todas las partes del árbol pueden ser movidas, borradas y agregadas, todo esto respetando las restricciones de XML. A diferencia de DOM donde hay secciones del árbol que no pueden ser cambiadas. Además, cuando se ha terminado de trabajar con el documento, JDOM permite serializarlo de regreso a un disco, o en un *stream* como una secuencia de *bytes*.