Journal of Geographic Information System, **2014, 6, 59-69**
Published Online February 2014 (http://www.scirp.org/journal/jgis)
http://dx.doi.org/10.4236/jgis.2014.61007

Transforming GML to Presentation Languages by Extending XSLT

Sajjad Hassany Pazoky, Farshad Hakimpour
University of Tehran, Tehran, Iran
Email: shpazooky@ut.ac.ir, fhakimpour@ut.ac.ir

Received December 22, 2013; revised January 22, 2014; accepted January 29, 2014

ABSTRACT

Diversity of practices and methods in all fields is the basis of emerging standards in different areas. World Wide Web Consortium (W3C) has standardized eXtensible Markup Language (XML) as a very convenient tool to structure data for numerous purposes. OGC standardized Geography Markup Language (GML), which is an XML-based language, to store and transport geospatial data. Despite the fact that it is a medium to separate geo-referenced data from presentation, GML by itself is not intended to visualize geo-referenced data. One of the solutions to visualize GML is to use eXtensible Style sheet Language Transformation (XSLT) as transformer to a visualization language such as Scalable Vector Graphic (SVG). Unlike the usual procedure, the major advantage of the proposed approach is that the transformation process is shifted to the client-side. XSLT as a median language is a general-purpose transformation tool. As it is not specialized for map cartography, map making process is very complicated using this primitive language. To facilitate transformation process, in this research, XSLT is extended to meet cartography requirements. Furthermore, a graphical user interface (XCartoT) is designed to set all the map properties interactively. XCartoT provides a user-friendly interface for cartographers to automatically generate necessary XSL files for their intended maps. The goal of this research is to develop a major step towards the geospatial Web.

KEYWORDS

Technology; Preference for Quality; Volume of Trade; Vertical Intra-Industry Trade

1. Introduction

The tendency towards geospatial services is incessantly growing among people from all different walks of life and becoming a non-detachable part of everyday life. People do not need high skills to use an online map whenever they enter a city for the first time. Furthermore, spatial mobile and Web services have raised the expectation of people. Thus, developing services to fulfill the requirements of all sorts of people from different background is an inevitable requirement that should be taken into account by geospatial specialists.

Based on high interest of the public to geospatial services, many specialists, organizations and institutes have developed various methods, practices, software, data formats, Web services, etc. in different fields of geospatial sciences. To prevent this diversity leading to confusion, standards play a pivotal role. The most dominant standardization organizations and consortiums in the field of geospatial Web services are International Organization of Standardization (ISO), World Wide Web Consortium (W3C) and Open Geospatial Consortium (OGC).

On the other hand, Web has undergone many revolutions compared to the existing one. The latest revolution was introduction of "Web 2.0" as a new concept. Web 2.0 is associated with concepts such as data/information sharing, interoperability, user-centered design and collaboration. Prior to this revolution, users were limited to use the available content, whereas Web 2.0 sites allowed the users to interact and collaborate in social networks with user-generated content.

On this account, previously known standards such as GML became more popular. GML is designed to transport and store geo-referenced data [1]. It separates content from graphic presentation and does not contain any information on how to visualize geo-referenced data. Therefore, GML is not directly suitable for applications that need visualization. Hence, it should be converted to other forms based on the usage such as graphical visualization, plain text or verbal output [2].

All the aforementioned possibilities can be implemented. However, graphical visualization is the most prominent way of communication. Geospatial science is meant to show the spatial relationships between natural features. Therefore, it is obvious that a graphical map is far more useful than a textual or verbal one.

GML can be visualized using certain libraries like Open Layers, certain GML viewer software, using XSLT to convert GML to a presentation language like SVG or X3D, etc. The focus of this research is on visualization of GML by converting it to SVG by XSLT. XSLT is used as the transformer of GML to SVG. Scalable Vector Graphics, a W3C recommendation since 2001, is a markup language for describing two-dimensional graphics applications and images, and a set of related graphics script interfaces. SVG is an XML-based language and describes vector graphic both statically and dynamically. As stated and discussed comprehensively in Peng & Zhang [3], the future of Web maps depends on SVG, GML and WFS. SVG has many features that make it the best choice for Web-maps including [4-7]:

- An open standard compatible with other Web standards such as CSS, XSLT, XLink, XPointer;
- Providing rich graphical visualization features such as scalability, vector display, animation, interactivity, transparency, graphic filter effects and special visual effects such as shadows, lighting effects and easy editing;
- Facilitating fast response time for interactive query requests;
- Providing efficient data interoperability over the networks;
- Supporting small wireless devices;
- Search ability.

SVG and GML are two complementary languages which are highly compatible and can work in synergy. Since GML plays a major role in geospatial Web, SVG is currently the most promising format for vector graphics display on the Internet [7,8].

Although this transformation is implemented several times, the research has improved it in the following ways:

- The transformation process is completely shifted to client-side.
- XSLT is a general-purpose tool to reorganize XML files andit does not have any special feature for map-making [9]. Imagine a cartographer intends to draw a legend for the map. Working with this primitive tool is more like drawing with a vector drawing software such as Auto Cad with the difference that user should write some lines of codes instead of drawing with drawing tools! In this research, XSLT is extended to meet map making requirements of a cartographer to convert GML to SVG. It is worth noting that the research is not to produce a GML viewer, but an advanced tool for a cartographer to convert GML to SVG.
- Since writing code in a somehow strange language like XSLT may be bothersome for non-specialists, a graphical user interface called XCartoT is designed and implemented to facilitate map making process for different classes of users. Using XCartoT, cartographers can set all the specifications of their desired map. XCartoT then generates XSLT file using extended functions. Choosing the GML file as the output, the transformation process is run and the resultant map in SVG format is displayed both textually and graphically.

Geospatial Web is a new term, emphasizing geo-referenced data anywhere and anytime. Mobile devices and the Internet are the major devices to accomplish this goal [10]. Therefore, the ultimate goal of this paper is to illustrate the essentiality for standardization organizations and institutes to force Web browsers to provide users with embedded capabilities to visualize geo-referenced data on the browsers both on computers and mobile devices.

2. Automatic Web Cartography

One of the major problems cartographers are facing today is the enormous amount of raw data. Thus, it is impossible to spend a long time working on every single sheet to produce a well-designed map. On the other hand, people are less likely to buy papermaps. The change in people's attitude from papermaps to digital maps is somehow a revolution.Development of telecommunication infrastructure such as wireless communication networks, spatial positioning methods such as Global Positioning Systems (GPS), radio frequency identification (RFID), gyroscope, mobile computing systems such as PDAs, tablets and smart phones has been the motive of this revolution [2]. Along with many other applications provided via telecommunication infrastructures and devices, people expect much more from maps, including availability and applicability in various devices anywhere and anytime. According to the definition of ubiquitous cartography, maps can be created and used anywhere and anytime. Hence, it is not possible to have highly-skilled crafts people spend long hours to produce general purpose maps, using traditional drawing methods like painting by hand [2]. Means of ubiquity are automatic and Web cartography which are discussed in the next section.

Automatic Web Cartography Techniques

Apart from technical issues, all Internet GIS services can be divided into two distinct categories based on the type of data used. The initial category of services uses cartographic data servers, while the other uses geodata servers.

Every Internet GIS service is constituted from data, logic and presentation elements [11]. The first category use data from a cartographic center, instead of raw geodata. Using this method, geodata is gathered, the map features are selected and designed to form a seamless and tiled map. Map providing procedure starts with a request from a user. Then, the raster tiles related to the request are searched and found in the cartographic database and sent to the user. Therefore, there is no logic behind these types of services. Although the method is very secure and lightens the workload of the client; it has also some serious disadvantages. The first disadvantage is that the response time may grow very much due to heavy load on the server. The second one is that the method needs high network bandwidth because the data transmitted via the network is usually very bulky. The architecture can only be implemented using server-side approach and is shown in **Figure 1(a)**.

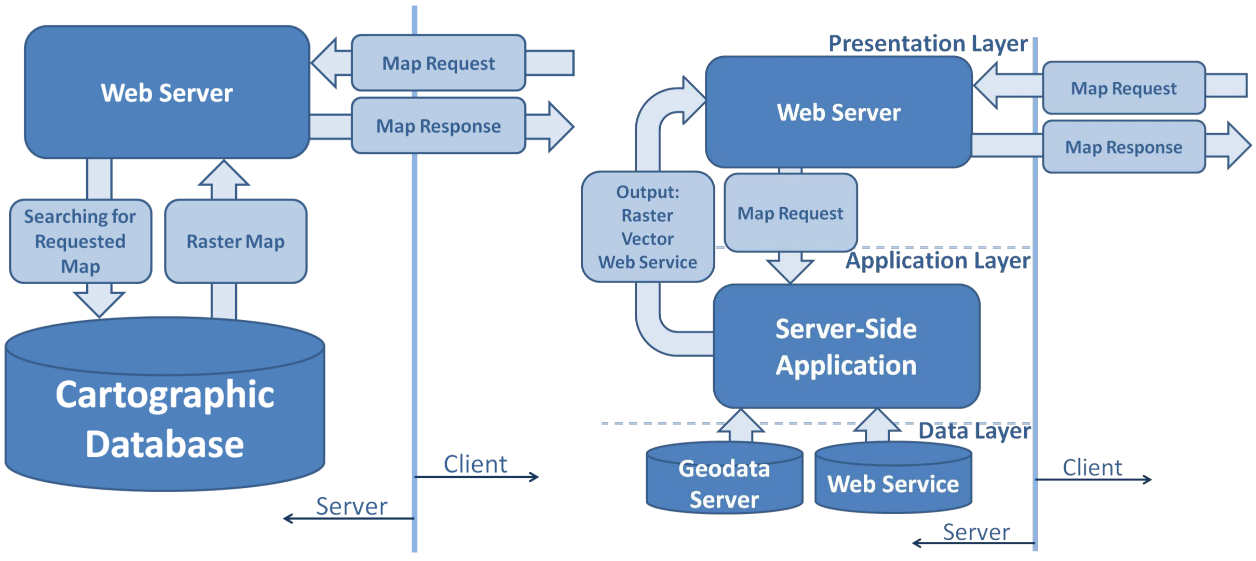
On the other hand, there are some other services that have direct access to geodata. Instead of previously-prepared raster tiles, the map is generated from raw geodata based on cartographic instructions provided by a specialist. These methods should be categorized into server-side and client-side approaches.

Server-side approaches have been implemented by two distinct technologies. The first technology is Common Gateway Interface (CGI) which is an interface for running external programs, software or gateways independent from platform. The most important bottleneck of this method is that if several CGI programs are operating simultaneously, it places considerable workload on the server. In addition, whenever a client makes any change to inputs, the CGI program starts a new process thread, consuming considerable computer resources [12]. The CGI-based approach is thus not a suitable method for map visualization on the Web [7]. Map Server is a software package that uses this approach. There are some codes for map presentation in the server-side. When the user sends a request, the map is generated and sent to the user as images [13,14].

Another server-side approach is using Java Servlets which is a program that runs on a Web server. Servlet usually works much faster than CGI, since it stays resident in memory when running. Another advantage of Java Servlets is its portability between operating systems and also servers [7,13]. The architecture of server-side methods is shown in **Figure 1(b)**.

The other alternative is to transfer some of the responsibilities of server to the client. There are two major methods for this approach.

The initial method is Java applet. A Java applet is a small program written in Java which runs on the client. Java applets are platform independent similar to Java Servlet. Running on the client side, developers can design user-friendly and interactive interfaces using Java



(a) (b)

Figure 1. How server-side Internet GIS services using (a) Cartographic database; (b) Geodatabase work.

applets. However, if the program and also the data are large, applet may overload the client and even paralyze the client machine [7].

Another solution is using plug-ins to install in client-side and add functionalities to the client's Web browser. Unlike Java products mentioned previously, plug-ins are browser and platform dependent. They provide additional abilities for the browser to display and process spatial data, so that the server's workload can be reduced. Since every computation required by every request is done on the requester machine, clients' interactions with the server can usually take place frequently and efficiently [7]. The architecture of the method is shown in Figure 2.

3. Proposed Approach

What shapes the foundation of this paper is that XSLT is not designed to produce visual maps from GML. It is a general-purpose language to reformat all types of XML documents. Therefore, making a map with all the ingredients using XSLT is very sophisticated. For example, in most cartography software, legend is generated automatically. But if users want to draw a complete legend on their own, it would be more similar to drawing in vector environment such as AutoCad or CorelDRAW. The difference is that instead of drawing tools provided in these environments, the user should write XSLT or SVG codes, which are very complicated and time-consuming. What makes the situation even worse is that since the graphical result of writing codes is not provided real-time, the process will become a trial and error process. Extending XSLT to meet cartographic requirements is the major idea presented in this paper. The paper provides functions, operators and ingredients for map cartography.

In this section, the investigator is to identify and describe the fundamental functionalities required to transform geo-referenced data to a visualization format such as SVG. In the next section, the functionalities are implemented using XSLT and it is clarified how Web cartography specialists can use them.

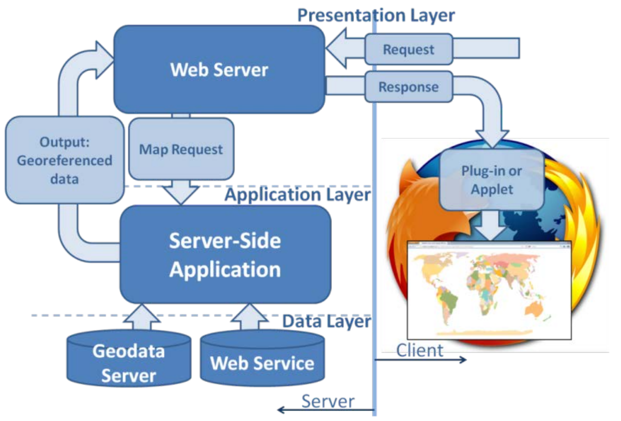


Figure 2. How client-side Internet GIS services using geodatabase work.

As discussed earlier, client-side computing has many advantages over server-side computing. In this paper, the process of converting GML to a presentation language like SVG is transferred to client-side using plug-in technology. Advantages of server-side approach are as follows:

- Higher security;
- Higher concealment;
- No need for the technologies to be widely approved.

On the other hand, advantages of client-side approach can be listed as follows:

- Less data is transferred on the network.
- Less frequently the data is transferred on the network.
- Higher level of interaction.

In addition, advantages of client-side approach for the discussed method are as follows:

- Separation of content from presentation;
- Reusability of XSLT file generated by the server;
- Possibility to make personal profiles for each client;
- Customizability of the resultant map both content-wise and graphically.

Another difference of this research with the previous ones is that XSLT is extended to visualize GML maps with SVG. Extending XSLT requires first to identify what needs to be added to current capabilities and then implement them.

The most straight forward way to extend XSLT is by extension functions. As a consequence, a set of functions with appropriate input and output are determined. The areas where functions should be considered are as follows:

- Introducing spaces;
- Transforming from real space to monitor space;
- Calculating scale;
- Drawing geospatial features with appropriate symbology and texts;
- Drawing outer map features.

The above-mentioned areas are necessary ones to extend XSLT in a way to be able to make an elementary, but complete map.

3.1. Introducing Spaces

The page that a viewer can see in a map is divided into several parts including the following:

- browser space;
- user space;
- map space;
- geospatial features space;
- outer features space.

The hierarchy of the spaces is depicted in Figure 3.

Functions providing the user with different spaces are mentioned below:

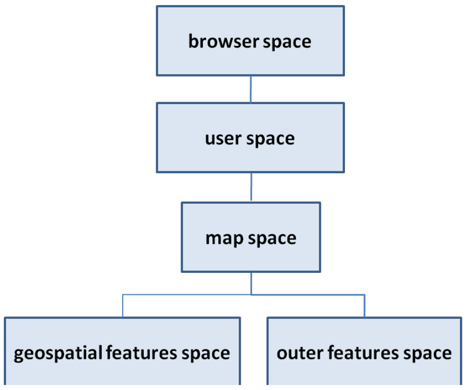


Figure 3. **Map spaces hierarchy.**

"*public static String view box* (*coords*, *margin*, *b Space Width*, *b Space Height*, *Precision*)"

The above function is responsible for calculating the minimum bounding box and receiving some global parameters from user with parameters as follows: *coords*: the coordinates of all the spatial features on the map; *margin*: browser space margin; *b Space Width*: width of the browser space; *b Space Height*: height of the browser space; and *Preciosion*: precision of all the numbers generated in the application.

"*public static String border* (*margin*)"

The above function creates the border of user space where *margin* is user space margin.

"*public static void margin Set* (*margin*, Umargin)"

The function creates the margin of map space using the following parameters: margin: margin between user space and geospatial features space; where *Umargin*: upper margin between user space and geospatial features space.

Another useful function of this category is to draw neatline. Neatline is a rectangle surrounding geospatial features space. The function is as follows:

"*public static String neatline* (*hShift*, *vShift*)"

Where *hShift* is horizontal shift of the geospatial features space; and *vShift* is vertical shift of the geospatial feature space.

3.2. Transforming Real Space to Monitor Space

There is a difference between coordinate system of real space and monitor space as shown in Figure 4.

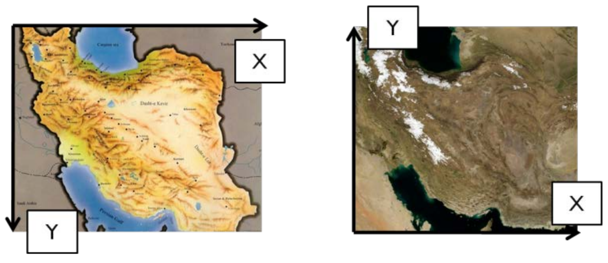
After calculating the appropriate transformation parameters between the two spaces, the function to perform the transformation by using the above equation is as follows:

"*private static String cords Change* (*coordPair*)"

Where *coordPair* shows coordinate pairs to be transformed from real space to monitor space.

3.3. Calculating Scale

To draw the map, the scale between real space and monitor space should becalculated. In order to achieve the goal, two elements are required: Space available on the monitor and actual ground space that accommodates real features. Space available on the screen as described above is asked from the user through "view box" function. Real space comprises of sets of nodes of all selected features. All the nodes in the GML file, which are intended to be drawn on the map, are also introduced to the "view box" function. Using maximum and minimum values of northing and easting, scales in vertical and horizontal directions are calculated using the following equations:



(a) (b)

Figure 4. **Difference of coordinates systems of (a) Monitor space; and (b) Real space.**

$$\text{horizontal scale} = \frac{w}{\max(X) - \min(X)}$$

$$\text{vertical scale} = \frac{h}{\max(Y) - \min(Y)}$$

where "*w*" is the width and "*h*" is the height of geospatial features space.

Based on the geospatial features space, scales in the two directions are different. To preserve the map from horizontal or vertical elongation, a unique scale should be chosen for both directions. As the geospatial features space introduced before cannot be exceeded, the minimum value of scales is chosen as the eventual map scale. An important point to mention is that choosing the minimum value of the two scales causes the geospatial features space to decrease in one direction. Thus, map designer should be alert not to leave any empty space.

The function to calculate the scale based on aforementioned equations is as follows:

"*private static double scale Detection* (*minE*, *maxE*, *minN*, *maxN*, *gfsBounds*)"

Parameters are as follows: *minE*: minimum easting; *maxE*: maximum easting; *minN*: minimum northing; *maxN*: maximum northing; and *gfsBounds*: geospatial feature space bounds.

3.4. Drawing Geospatial Features with Appropriate Symbology and Text

Geospatial features are categorized into three groups: points, polylines, and polygons. For each of the groups, drawing, labeling and symbology is different. Therefore, different functions should be considered for each category.

- Points: Points are comprised of just two coordinate values of easting and northing. Point symbols should be defined in SVG file and the function refers to the symbol. Furthermore, the size of the symbol is adjustable using the function. The function to draw points with appropriate symbology is as follows:
  "*public static String stylePoint* (*coordPairs*, *sym*, *fName*, *labelTag*, *symH*, *symW*, *labelStyle*)"
  Using the following parameters: *coordPairs*: coordinate pairs representing a point; *sym*: reference to SVG point symbol; *fName*: name of the feature group in the legend; *labelTag*: tag to label features accordingly; *symH*: symbol height; *symW*: symbol width; and *labelStyle*: label styling.
- Polylines: Polylines are comprised of two sets of coordinates. As there is no way to define linear symbols in SVG, polyline symbols are implemented using "style" which is a CSS reserved keyword. The text of a linear feature should be written on it several times with a certain interval. The function for drawing polylines with appropriate styling is as follows:
  "*public static String stylePolyline* (*coordPairs*, *fName*, *labelTag*, *labelInt*, *styling*, *labelstyle*)"
  Parameters are as follows: *coordPairs*: sets of coordinate pairs representing a line; *fName*: name of the feature group in the legend; *labelTag*: tag to label features accordingly; *labelInt*: labels interval; *styling*: defining linear symbols using CSS code; and *labelStyle*: label styling.
- Polygons: Polygons are also comprised of two sets of coordinates. Like points, polygons can use externally defined symbols. In addition, as polygons are surrounded with lines, border lines can be styled by CSS "style" keyword. Labels of polygons should be in the middle of the polygon. The function for drawing polygons is as follows:
  "*public static String stylePolygon* (*coordPairs*, *sym*, *fName*, *labelTag*, *styling*, *labelStyle*)"
  With the following parameters: *coordPairs*: sets of coordinate pairs representing a polygon; *sym*: reference to SVG polygon symbol; *fName*: name of the feature group in the legend; *labelTag*: tag to label features accordingly; *styling*: defining linear symbols using CSS code for polygon borders; and *labelStyle*: label styling

3.5. Drawing Outer Map Features

Outer map features are those which help map users understand the map better and include the followings:

- Map title: Map title is a short hugely-typed text on top of a map to provide a clear indication on what the map is displaying. Therefore, top of the map should be left empty that is accomplished by setting different margin value for top of the map when using "marginSet" function. The function designed for producing map title should suggest the best place for title. Also, two parameters are needed that if the suggested position is not acceptable for the users, they can move it in two directions. The function which displays the title is as follows:
  "*public static String title* (*text*, *hOffset*, *vOffset*, *styling*)"
  Parameters are as follows: *text*: title text; *hOffset*: horizontal offset of the title; *vOffset*: vertical offset of the title; and *styling*: title text styling.
- Grid: Grid lines define locations on map using Cartesian coordinate system. Each grid line specifies one coordinate horizontally or vertically based on its direction. The function is as follows:
  public static String grid (*startingE*, *startingN*, *hInt*, *vInt*, *dist*1, *hOffText*, *vOffText*, *textStyle*)
  Using the following parameters: *startingE*: the easting of the first vertical grid line; *startingN*: the northing of the first horizontal grid line; *hInt*: interval between horizontal grid lines; *vInt*: interval between vertical grid lines; *dist*1: distance of grid line texts from grid lines; *hOffText*: horizontal offset of texts; *vOffText*: vertical offset of texts; *textStyle*: texts styling.
  Different parameters of grid function are depicted in **Figure 5(a)**.
- Legend: Legend is where all symbols used in the map are gathered showing which feature they refer to. The function declaration should be as mentioned below:
  "*public static String drawLegend* (*pX*, *pY*, *width*, *titleStyle*, *textStyle*, *borderStyle*)"
  With the following parameters: *pX*: X coordinate of the position of the legend in outer features space; *pY*: Y coordinate of the position of the legend in outer features space; *width*: legend width; *titleStyle*: styling of the titles of the legend; *textStyle*: styling of the text used in the legend; and *borderStyle*: styling of the border of the legend.
  Different Parameters of legend function is depicted in **Figure 5(b)**.
- Scalebar: Scalebar is a graphical representation of scale that helps users measure distances on a map. Since scalebar is more complicated than other outer map features and has more distinct elements, parameters of this function are more than previous ones. The function declaration is as follows:
  "*public static String scaleBar* (*type*, *div*, *divLength*, *subDiv*, *pX*, *pY*, *dist*1, *dist*2, *height*, *titleStyle*, *headStyle*, *oddStyle*, *evenStyle*, *titleHOff headHOff*)"
  Parameters are as follows: *type*: scalebar type; *div*: number of divisions; *divLength*: division length;

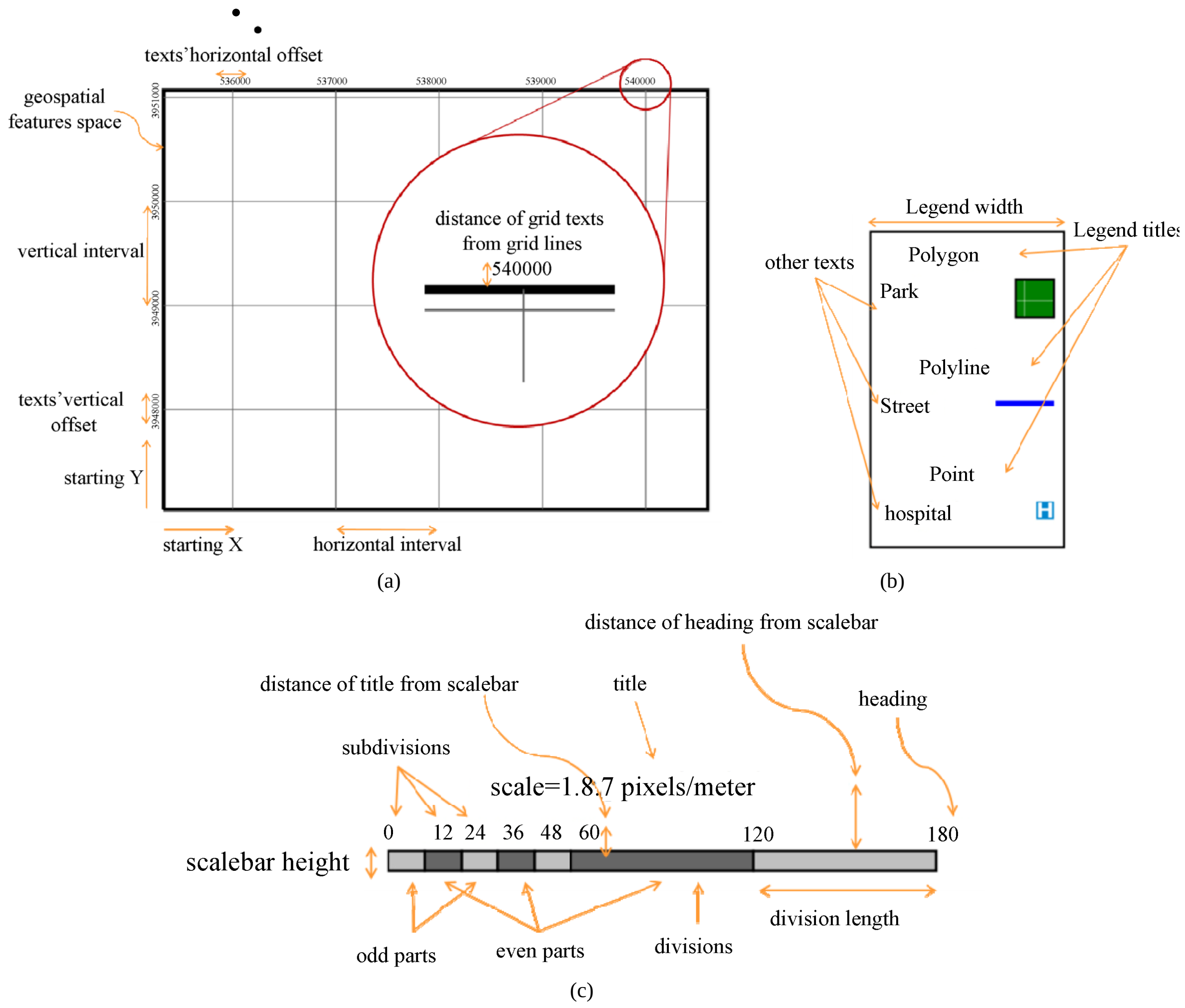


Figure 5. Different parameters of (a) "grid"; (b) "drawLegend"; (c) "scalebar" functions.

subDiv: number of subdivisions; *pX*: X coordinate of the position of the scalebar in outer features space; *pY*: Y coordinate of the position of the scalebar in outer features space; *dist*1: distance of title from scalebar; *dist*2: distance of heading from scalebar; *height*: scalebar height; *titleStyle*: title styling; *headStyle*: heading styling; *oddStyle*: scalebar styling odd parts; *evenStyle*: scalebar styling even parts; *titleHOff*: title horizontal offset; and *headHOff*: heading horizontal offset.

Different Parameters of scalebar function is depicted in Figure 5(c).

- North arrow: North arrow type is defined by symbols in SVG directly and referenced in the function. The function is as follows:
  "*public static northArrow* (*pX*, *pY*, *sym*)"
  Using the following parameters: *pX*: X coordinate of the position of the north arrow in outer features space; *pY*: Y coordinate of the position of the north arrow in outer features space; and *sym*: reference to the symbol.
- Map metadata: Every map should contain production information, otherwise it is not valid. The text is entered by the user and '\n' wild card should be used to go to next line. The function is as follows:
  "*public static metadata* (*pX*, *pY*, *text*, *textStyle*)"
  With the following parameters: *pX*: X coordinate of the position of the north arrow in outer features space; *pY*: Y coordinate of the position of the north arrow in outer features space; *text*: metadata text; and *textStyle*: Text Styling.

4. Prototype Implementation

Required functions to extend general-purpose XML Transformation language to a cartographic tool were introduced in the previous section. In this section, most of the functions are implemented to demonstrate how using extension functions can facilitate GML to SVG conversion on the client side with conditions described in previous sections. Furthermore, a graphical user interface called XCartoT is designed to make the process more user-friendly and also allow less professional users to produce well-designed maps using aforementioned tools.

4.1. Implementing Extension Functions

4.1.1. Data

The data used in the research consists of the following:

- Polygon features: parcels of 6 municipal regions;
- Polyline features: streets of the same regions;
- Point features: petrol stations, police stations and hospitals of the same regions.

4.1.2. XSLT File

The first line of code to talk about is the opening <svg> tag that is as follows:

'<svg height="" width=" " viewBox="" version="1.1" preserveAspectRatio="none">'

In this line, 3 parameters should be set. The height and the width of the page which specifies browser space should be entered by the user. The value of the "viewBox" attribute is a list of four numbers <min-x>, <min-y>, <width> and <height>, separated by whitespace and/or a comma. This attribute determines a rectangle in user space which should be mapped to the bounds of the viewport established by the given element. It is noteworthy to mention that this attribute is mandatory and cannot be left empty [4]. "viewbox" function described in the previous section is used to calculate the value of this attribute. This line of code runs "viewbox" function and sets the output to a "viewbox" variable:

'<xsl:variable name="viewbox" select="carto:viewbox (//gml:posList,5,1260,600,2)" />'

Where '//gml:posList' is an XPath expression inputting the coordinates of all spatial features.

After substituting values, SVG tag would be as follows:

'<svg height="600" width="1260" viewBox= "{$viewbox}" version="1.1" preserveAspectRatio = "none">'

Where "{$viewbox}" is the name of the variable to store the output of "viewbox" function.

Next step is to call the "border", "marginSet" and "neatline" functions to determine the coordinates of different spaces which are depicted in **Figure 6(a)**. The functions may be called as follows:

'<xsl:variable name="border" select="carto:border (10)" />'

'<xsl:value-of select="carto:marginSet (20,50)" />'

'<xsl:variable name="neatline" select="carto:neatline (270,0)" />'

The user is also able to draw map border and neatline. To do so, the result of these functions should be used to draw rectangles using the SVG code lines below:

'<polyline points="{$border}" style="fill:white;stroke:black;stroke-width:2" />'

'<polyline points="{$neatline}" style="fill:white;stroke:black;stroke-width:1" />'

To draw geospatial features in the provided geospatial feature space, different symbologies can be defined using SVG capabilities which some samples are shown in **Figures 6-8**.

In the XSLT file, geospatial features are drawn one by one. Thus, drawing should be inside a loop.

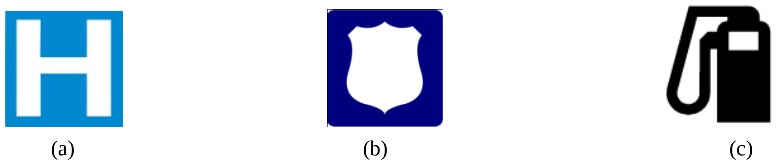


Figure 6. **Sample point symbols for (a) Hospital; (b) Police department; and (c) Petrol stations.**

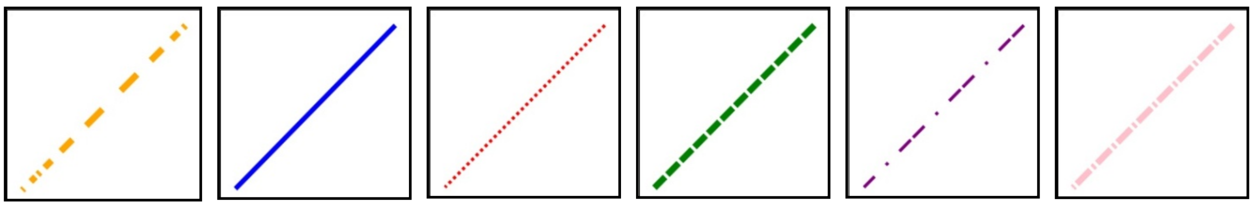


Figure 7. **Sample linear symbols.**

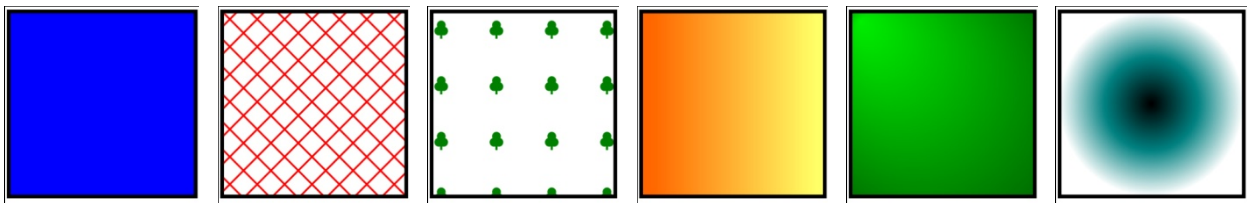


Figure 8. **Sample polygon symbols.**

At this stage, outer map features are added to the map. The ones which are implemented in this research are as follows:

- map title
- grid
- legend
- scalebar

"title" function is called as follows:

'<xsl:value-of select="carto:title('Sample Map',-350, 25,'font-family:Vernada;font-size:24pt')"/>'

The following line of code is to produce grid lines:

'<xsl:value-of select="carto:grid(536000,3948000,1000, 1000,5,'stroke:rgb(99,99,99);stroke-width:1','font-family :Vernada;font-size:7pt',-15,-15)" />'

The following line of code produces a legend:

'<xsl:value-of select="carto:drawLegend(1050,50,130, 'font-family:Vernada;font-size:10pt','font-family:Vernada ;font-size:7pt','fill:white;stroke:black;stroke-width:1')" />'

And finally, the following line of code is used to produce a legend:

'<xsl:value-of select="carto:scaleBar(1,3,600,5,990, 70,20,5,8,'font-family:Vernada;font-size:12pt','font-famil y:Vernada;font-size:7pt','fill:#C0C0C0;stroke:black;stro ke- width:1','fill:#9ACD32;stroke:black;stroke-width:1',- 80,-2)" />'

The final result with all the map surroundings is as depicted in **Figure 9**.

4.2. Developing a GUI for the Application (XCartoT)

Using extension functions to facilitate map making from geo-referenced data might confuse users unfamiliar with geospatial and Web concepts. In other words, Web car-tographers may find it difficult to learn and use XSLT language as well as our extended functionalities. To alleviate this problem and simplify map making process, a user-friendly graphical user interface called "XCartoT" is designed. XCartoT generates the necessary XSLT file automatically according to the cartographers' requirements. In this section, XCartoT is introduced. First view of XCartoT is shown in **Figure 10**.

XCartoT is made up of two parts: The menu bar and some tabs. The hierarchy of the tabs is shown in **Figure 11**.

In 'Map Generation Features' tab, user should set all the parameters required to produce a map. The parameters are those described in Sections 5 and 6. For better organization, these parameters are categorized into four groups: Basic Parameters, Drawing Layers, Symbology and Outer Map Features.

After setting all the parameters, XSLT file is produced automatically using extension functions described in previous sections. Going to "Code" tab and choosing XSLT nested tab, the user can view automatically-generated XSLT file.

"GML" tab shows GML file after being introduced to the application. GML fileshould be added to the application using the menu bar. Having GML and XSLT files, SVG file is produced by choosing "Transform" menu item. Then, it is visualized by ajava SVG visualization library called Batik and can be viewed in "SVG View" tab as shown in **Figure 12**. Finally, the whole procedure can be summarized in the following flowchart (**Figure 13**).

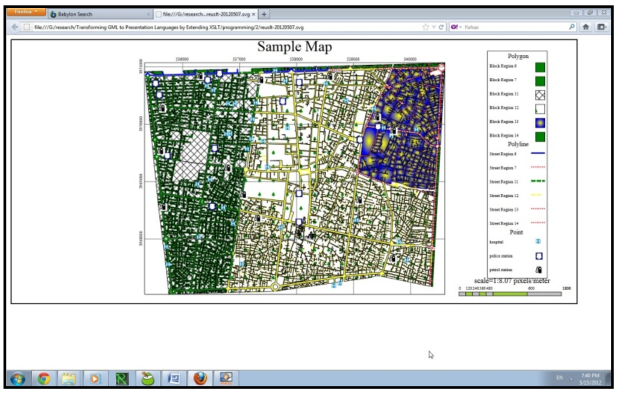
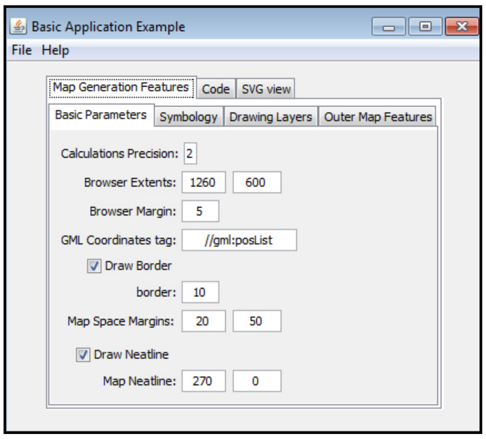


Figure 9. **Result map after adding geospatial features with their symbology.**


Basic Application Example
File Help
Map Generation Features | Code | SVG view
Basic Parameters | Symbology | Drawing Layers | Outer Map Features

Calculations Precision: 2
Browser Extents: 1260 600
Browser Margin: 5
GML Coordinates tag: //gml:posList
☑ Draw Border
border: 10
Map Space Margins: 20 50
☑ Draw Neatline
Map Neatline: 270 0

Figure 10. **First view of the GUI.**

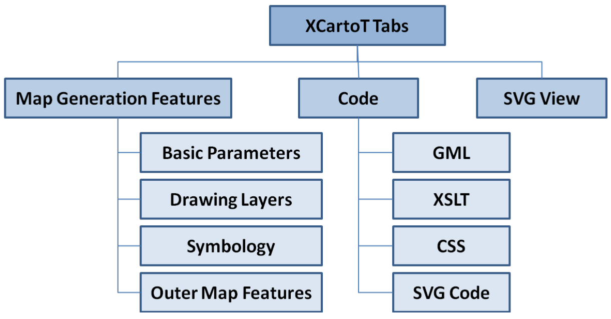


Figure 11. **The hierarchy of XCartoT tabs.**

5. Literature Review

This section is dedicated to introduce and compare work of various other researchers in this area. The first research was done by Mansfield, P and Fuller, D. [15]. The authors introduce the idea of converting XML to SVG via XSLT. XML data used in the research is not geospatial and there's no example of converting geospatial data to SVG.

Another research is done by Taladoire, G. [16]. The author has explained the problem well and described all the details. The input data used is GML, but a complete conversion from GML to SVG is not shown. The author has used CSS for styling, but the resultant map is not interactive.

Another research is done by Neumann, A. and Winter, A.M. [17]. The authors demonstrate that vector graphic has higher quality than raster graphic. Then, languages for vector graphics are introduced and compared. These languages include: SVF, DWF, Flash, PDF, PGML, WebCGM, HGML, DrawML, VML, Java2D and XML. The conclusion of the comparison leads to the fact that SVG is the best choice to present the maps on the Web. The major advantage of this research is that the maps are completely interactive.

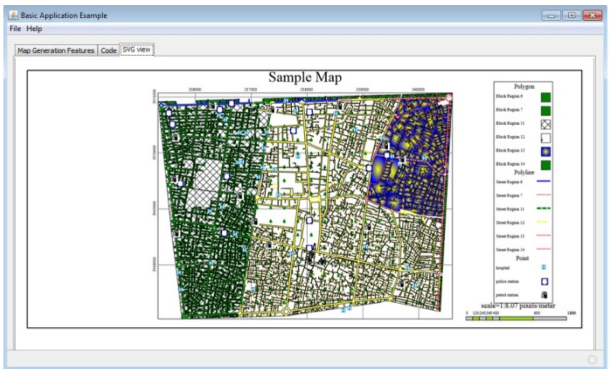


Figure 12. SVG view generated after producing SVG code in "SVG View" tab.

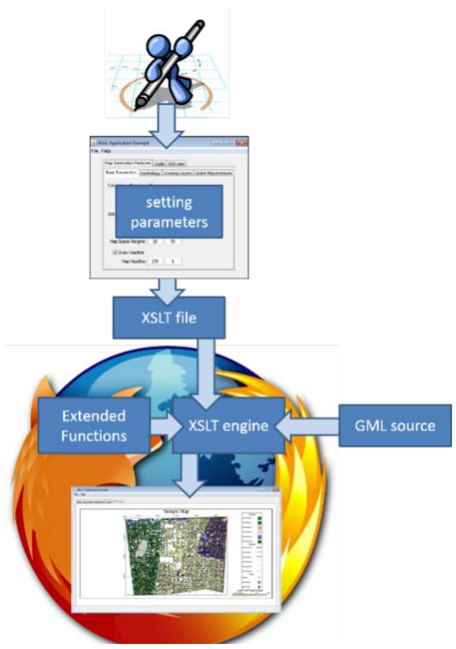


Figure 13. The whole procedure of map production using the proposed method.

A very efficient conversion of GML to SVG is done by Tennakoon, W. [18]. The author has converted a complete GML file to SVG using XSLT code.

Another attempt is done by Spanaki *et al.* [19] in Athens, Greece. The difference between this paper and the previous attempts is that the conversion is done in the internet browser of the client.

Bonati *et al.* [20] have also researched in this area. Their approach is exactly similar to the previously cited papers, but the generated map is completely interactive. The user can pan, zoom, and change symbols and much more.

A paper assembled by Ron in Galdos Systems Inc. is somehow similar to the idea mentioned in this research. The difference between this research and previously cited papers is that the author introduces the idea of extension functions in XSLT. This function should be implemented in the XSLT engine to work as is done in this research. However, there is no implementation of extension functions in this research.

6. Conclusions

Emerging various methods, practices, software, etc has caused standards to be the necessities of today's life. GML as a standard, which is recommended by OGC, is a language to store and to transport geospatial data. Having no information on how to visualize the data graphically, GML is a means to separate geospatial content from graphical presentation. Therefore, there are several ways to visualize the textual geo-referenced data. A straightforward method is to convert GML to a graphical language like SVG, using a convertor like XSLT. Although the process has been implemented several times by different scientists, the research has improved the current approaches inseveral aspects:

- Transformation process is completely transferred to client-side. Using the proposed approach, the user has access to geodata, as well as an appropriate visualization of the data. Another advantage is that geodata transferson the network much faster than raster images. Furthermore, geodata is transferred once and can be visualized many times without any need to connect to the server. Unlike server-side approaches, there is not much load on the server, since it just prepares and sends geo-referenced data and sometimes XSLT file needed for presentation. The next advantage is that there is no need to install any special software on the computer or mobile device except fora mere internet browser of any vendor because the required plug-ins are pre-installed on all major internet browsers, although the quality of the resultant maps might have slight differences in special conditions such as animation.

- The next important contribution of the paper is that XSLT as a general-purpose transformation language is extended to meet cartographic requirements of map-making. Required functions are recognized, designed and implemented. Although several functions can be thought of to be added to XSLT, the research focuses on the most important ones.
- The final contribution is developing a GUI called XCartoT to further facilitate users in map-making process through this method. XCartoT allows users to set all the parameters of a map and the XSLT file is generated automatically using extension functions. Then, after introduction of GML file, conversion is executed and the resultant SVG file is available for the user both textually and graphically.
- Geospatial Web is a concept in information era to provide access to maps anywhere and anytime using the Internet and mobile devices. The goal of the research is to prepare the required steps towards the geospatial Web. Furthermore, the investigator suggests that standardization organizations and consortiums such as W3C oblige Web browsers for both for computers and mobile devices to support geospatial services.

REFERENCES

[1] Open Geospatial Consortium (OGC), "Geograpy Markup Language (GML)," 2007. http://www.opengeospatial.org/standards/gml

[2] G. Gartner, D. A. Bennet and T. Moritta, 'Towards Ubiquitous Cartography," *Cartography and Cartographic Information Science*, Vol. 34, No. 4, 2007, pp. 247-257. http://dx.doi.org/10.1559/152304007782382963

[3] Z. Peng and C. Zhang, "The Roles of Geography Markup Language (GML), Scalable Vector Graphic (SVG), and Web Feature Service (WFS) Specifications in the Development of Internet Geographic Information Systems (GIS)," *Journal of Geographic Systems*, Vol. 6, No. 2, 2004, pp. 95-116. http://dx.doi.org/10.1007/s10109-004-0129-0

[4] World Wide Web Consortium (W3C), "Scalable Vector Graphics (SVG) 1.1 (Second Edition)," 2011. http://www.w3.org/TR/SVG/

[5] A. Neumann and A. M. Winter, "Time for SVG—Towards High Quality Interactive Web-Maps," *Proceedings of the* 20*th International Cartographic Conference*, Beijing, 6-10 August 2001, pp. 2349-2362.

[6] B. Jenny, A. Terribilini, H. Jenny, R. Gogu, L. Hurni and V. Dietrich, "Modular Web-Based Atlas Information Systems," *Cartographica*, Vol. 41, No. 3, 2006, pp. 247-256. http://dx.doi.org/10.3138/2773-Q553-0483-NP77

[7] X. Yao and L. Zou, "Interoperable Internet Mapping: An Open Source Approach," *Cartography and Geographic Information Science*, Vol. 35, No. 4, 2008, pp. 279-293. http://dx.doi.org/10.1559/152304008786140560

[8] M. Kramis, C. Gabathuler, S. I. Fabrikant and M. Waldovogel, "An XML-Based Infrastructure to Enhance Collaborative Geographic Visual Analytics," *Cartography and Geographic Information Science*, Vol. 36, No. 3, 2009, pp. 281-293. http://dx.doi.org/10.1559/152304009788988305

[9] World Wide Web Consortium (W3C), "XSL Transformations (XSLT) Version 2.0," 2007. http://www.w3.org/TR/xslt20/

[10] A. Scharl and K. Tochtermann, "The Geospatial Web: How Geobrowsers, Social Software and the Web 2.0 Are Shaping the Network Society," Springer, London, 2007. http://dx.doi.org/10.1007/978-1-84628-827-2

[11] Z. R. Peng and M. H. Tsou, "Internet GIS," Wiley, Hoboken, 2003.

[12] S. Gundavaram, "CGI Programming on the World Wide Web," O'Reilly Associates, Sebastopol, 1996.

[13] B. Huang and M. F. Worboys, "Dynamic Modelling and Visualization on the Internet," *Transactions in GIS*, Vol. 5, No. 2, 2001, pp. 131-139. http://dx.doi.org/10.1111/1467-9671.00072

[14] B. Kropla, "Beginning MapServer: Open Source GIS Development," Springer-Verlag, New York, 2005.

[15] P. Mansfield and D. Filler, "Graphical Stylesheets Using XSLT to Generate SVG," *Proceedings of XML Conference*.

[16] G. Taladoire, "Geospatial Data Intergration and Visualisation Using Open Standard," *7th EC-GI & GIS Workshop*, Potsdam, 13-15 June 2001.

[17] W. T. M. S. B. Tennakoon, "Visualization of GML Data Using XSLT," Diploma Thesis, International Institute for Geo-Information Science and Earth Observation, Enschede, 2003.

[18] M. Spanaki, B. Antoniou and L. Tsoulos, "Web Mapping and XML Technologies: A Close Relationship," *7th AGILE Conference on Geographic Information Science* Heraklion, 29 April-1 May 2004.

[19] L. P. Bonati, L. Fortunati and G. Fresta, "SVG Explorer of GML Data," *Proceedings of SVG Open*, Vancouver, 13-18 July 2003.

[20] R. Lake, "Making Maps for the Web with GML," Galdos Systems, Inc.