USING XML LANGUAGES FOR MODELING AND WEB-VISUALIZATION OF GEOGRAPHICAL LEGACY DATA

Brigitte Mathiak, Andreas Kupfer and Karl Neumann
Institut für Informationssysteme - TU Braunschweig
Mühlenpfordtstr. 23, 38100 Braunschweig, Germany
{mathiak, kupfer, neumann}@idb.cs.tu-bs.de

Abstract In our aim to modernize geographical legacy data from the German office of geographical survey with XML languages, we first modeled the data in GML, a standard language for geography markup. The resulting model has then been used as a template for a JAVA application that assembles the necessary information from the legacy data and writes it into a GML document. During the second part of our work we mapped GML with the extensible style-sheet language for transformation (XSLT) into scalable vector graphics (SVG). The resulting SVG graphics share close resemblance to official maps.

Keywords: GML, SVG, XSLT, geographical data, XML

1. Introduction

For the exchange of geographical data and its computer-based usage the Internet becomes more and more important (Kraak and Brown, 2000). In this context special techniques for the unification of the geographical data formats are necessary to provide means for the format-independent exchange of the data (Fidalgo et al., 2003). XML (Harold and Means, 2002; Goldfarb and Prescod, 2003) gains more and more importance in the field of Internet data exchange. Therefore it is quite obvious to apply XML to geographical data as well.

XML provides the base for GML (geography markup language), a language specified by the Open Geospatial Consortium (OGC, 2001), an international consortium of about 200 companies and organizations. GML was designed as an open standard to permit the exchange of geographical data without establishing too many constraints on how the data is structured. GML works with XML Schema (E. van der Vlist, 2003), defining several schema types which can be used or derived in a customized model. It also provides a theoretical

framework for the structure, yet leaves enough individuality to fit almost every given data set.

Our goal is to improve the usability of data from the German office of geographical survey by using XML. This legacy data has not been structurally changed for over a decade. Thus making it a challenging starting point for a GML model. Additionally to the model definition we also implemented a JAVA application which transforms the legacy data to the GML model. As a further step to demonstrate the improved usability, we decided to attempt a visualization (Kraak and Ormeling, 2002) of GML by using even more XML languages. The obvious choice for the graphical description language was SVG (scalable vector graphics) (W3C, 2000). For the transformation from GML to SVG (Guo et al., 2003) we used a third XML language: extensible style sheet language for transformation (XSLT) (Kay, 2001; W3C, 1999; Bex et al., 2002). XSLT documents contain instructions for the transformation of XML documents into differently structured XML documents. We use the open source XSLT processor Xalan (Apache, 2003) for transformation, but any XSLT-conform processor will do as well. Cascading style sheets (CSS) (Meyer, 2002) control the style of the various cartographic objects. Adjustments to the map's style can be easily made by either changing the CSS or in more complicated cases changing the standardized XSLT templates.

This paper gives a short overview over the mentioned XML languages and the structure of the geographical legacy data. We also discuss the applications of the languages in the transformation process. In section 2 we describe GML, our customized GML model and how we transformed the legacy data into it. Sections 3 and 4 give introductions to SVG and XSLT and how we used them to produce map-like graphics from the GML data. Section 5 contains a summary of the whole process.

2. Geography Markup Language

GML or Geography Markup Language is an XML based encoding standard for geographic information developed by the Open Geospatial Consortium (OGC). The main elements of the GML modeling structure are geographic features, which represent real world geographical objects. Their geometric properties are modeled by a uniform geometric definition using two-dimensional coordinates. The current version of GML is GML 3.1 (OGC, 2004), though the version used in this case study is GML 2.0 (OGC, 2001). GML 2.0 is defined in XML Schema (XSD) and is divided into three parts for structuring purposes: Geometry schema (geometry.xsd), feature schema (feature.xsd) and XLinks schema (xlinks.xsd) to provide support for linking the geographical data with other sources and to allow referencing on GML-documents. All three schemas are provided by the OGC.

Geometries and Features in GML

The geometric properties of GML are strictly defined. There are five geometric types, all derived from one abstract GML geometry type. First, ”Point” contains one pair of coordinates, while ”Line” contains more pairs which are connected by straight lines. ”Box” is defined by its lower left corner and upper right corner. ”Linear Ring” is a closed line circuit of at least three different points, additionally the last point must be the same as the first point, so all in all there have to be at least four points. In contrast to ”Linear Ring” a ”Polygon” in GML may possess holes, inner boundaries that mark areas that are cut out from the area of the outer boundary. A ”Polygon” therefore consists of one outer boundary and possibly one or more inner boundaries.

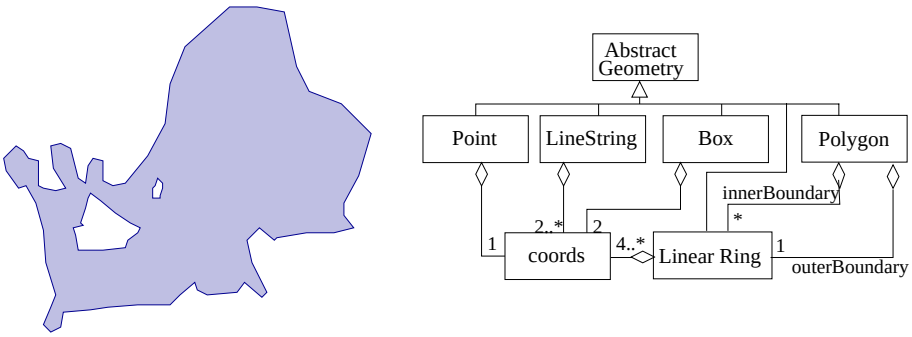

Figure 1. Polygon with holes and the UML schema of the GML geometry

While the form of the geometrical description is very exactly defined, the form of the actual features is not. Instead GML offers a structural framework to fit the scenario it is meant to describe. Every model is made up of features. These features have properties. The properties have a name, a type and a value. The structure of the feature, hence the number and types of the properties, is defined by its type. A feature collection is also considered a feature. The contained features are the values of the collection’s property. The contained features may contain other features thereby building a hierarchy.

The actually used feature types are user-defined in a XML Schema. The advantage of XML Schema is, that it allows object-oriented handling of the types like derivation or the combination of several types into one substitution group. GML does state some rules that are not directly given by the schemata like naming conventions and the obligatory derivation from the base types. These rules are additional to the ones given by the schemata. GML allows the users to model their own version of GML, fitting their individual needs. This individuality is much needed as geographical data is not very standard-

Table 1. Extractions of the respective files corresponding to the object "Bachstr."

file name	**lines containing 86118065**
objecttype.txt	86118065, 3101 *(means street)*
name.txt	86118065, Bachstr.
coordinates.txt	86118065, 4437960.070, 5331818.450, 4437967.200, 5331825.410 86118065, 4437952.980, 5331812.550, 4437960.070, 5331818.450

ized and may contain very different kinds of meta-information. The conform appearance of the data, especially the geometric data, does allow some degree of interchangeability with other GML models, yet it is still general enough to model most geographic scenarios.

Modeling landscape data with GML

Since GML already models the geography and the base feature outline, the main focus for modeling lies in the definition of new, customized feature types and their relationships. In order to customize the feature types best, we must first have a look at the form of the legacy data.

The so-called Digital Landscape Model (DLM) is part of a German project, which has the purpose of digitizing the surface in the Federal Republic of Germany topographically and cartographically. It has now been running for over 15 years with great success, yet most of the specification for the data has been made in the initial phase of the project, hence 15 years ago, with no systematical changes since that time. The landscape data for example has a catalogue for object types that encodes the object types into four-digit numbers. So the dataset for a street does not contain the word "street", but instead the code 3101 which can be decoded into "street" using the non-database catalogue. There are about 200 object types and over 1000 attributes, which contain additional information for the object types, like "width of street" for a street or "kind of vegetation" for a forest. The attributes are encoded in a similar way as the object types using a three-letter code for the attribute type and four digits for its value.

The original DLM data can be downloaded from the Internet for test purposes. An additional program, EDBS-Extra (Rinner, 2001), is used to transform the original format into a more accessible format, which uses a quasi relational framework for the data. The transformed information is stored in the file corresponding to the type of information that is extracted, all linked by the ID of the object that the information belongs to (see an example in Table 1).

In order to model this data structure in terms of GML a complex type called DLMMemberType is introduced, of which the object types like ”street” are to be instances. The objects themselves are modeled as features, therefore DLM-MemberType is inherited from AbstractFeatureType the GML base type for all features. The DLMMemberType should be able to contain all available data from the DLM to avoid information loss.

All DLM objects have an object ID, a name, attributes and some kind of geometry. DLM geometric data is either point, line or polygon in shape. This corresponds to the GML geometries ”Point”, ”Line” and ”Polygon” or more precisely to the GML geometry-containing elements ”location”, ”center-LineOf” and ”polygonMember”. In the following example a simplified version of the DLMMemberType is shown on the left side as it is defined in the XML Schema definition for the DLM model. The right side is a corresponding sample feature, derived from the above example. As one can see, the DLMMemberType is defined as a complex type with a complex content extended from AbstractFeatureType. Note that DLMMemberType and AbstractFeatureType do not share the same namespace, since AbstractFeatureType is part of GML, while DLMMemberType belongs to the DLM model. In line 4 a sequence of elements begins. Every DLM object has an ID, a name, a geometry and some attributes which are modeled in the corresponding GML features as elements in the lines 4-12. The three possibilities for the geometry are modeled by the choice in the lines 6-10.

```
 1 <complexType name="DLMMemberType">
 2  <complexContent>
 3   <extension base= "gml:AbstractFeatureType">
 4    <sequence>
 5     <element ref="objectID" />
 6     <choice> <!-- geometry-->
 7      <element ref="gml:location"/>
 8      <element ref="gml:centerLineOf"/>
 9      <element ref="gml:polygonMember"/>
10     </choice>
11     <element ref="attributes" />
12    </sequence>
13   </extension>
14  </complexContent>
15 </complexType>
```

The ”DLMMemberType” is designed to fit all the DLM object types, like ”Street”, ”Sea”, ”Tower” or ”Border” and was constructed by modeling the structure from the DLM catalogue. Some other types were defined as well eg. ”DLMModel” the root element, which are further explained in the next section.

Transforming landscape data

For the transformation process we decided to use JAVA, since it allows easy XML Schema parsing and validation through the Xerces parser (Apache, 2002) and easy handling of regular expressions for the DLM parsing. In order to transform the original DLM data to GML several adjustments have to be made. Object types and attribute names and values are decoded using a list extracted from the DLM catalogue, which was hand-adjusted for fitting XML and GML naming conventions (no metacharacters, no spaces). The list is also transformed into an additional schema to define all the different object types and DLM attributes so they can be used in the GML document.

Below is the example of a very small ”DLMModel”, only containing one example street. The DLMModel element (line 1) is the root element of the GML document and would normally contain all the namespace attributes and the schema location, which are too long to be listed here. It also contains one or more ”dlmMember” elements (line 2), that are a wrapper elements for the features. The wrapper elements are predetermined by GML to define the association between the model and the feature e.g. it may include an XLink to a remote feature instead of a regularly included feature.

The example feature ”Street” is an element of type ”DLMMemberType”. The element ”gml:name” in line 4 is inherited from the ”AbstractFeatureType” and therefore had not to be redeclared in the schema definition in section 1. The attributes (lines 20-26) are decoded in human language and the coordinates (lines 7-18) are assembled from the line pieces in table 1 to a connected line string. The element ”gml:centerLineOf” in line 6 is a wrapper element for the element ”gml:LineString” and works similar to the ”dlmMember” except that it contains a geometric line element instead of a feature element.

```
 1<DLMModel>
 2 <dlmMember>
 3  <Street>
 4   <gml:name> Bachstr. </gml:name>
 5   <objectID> 86118065 </objectID>
 6   <gml:centerLineOf><gml:LineString>
 7    <gml:coord>
 8     <gml:X> 4437952.980 </gml:X>
 9     <gml:Y> 5331812.550 </gml:Y>
10    </gml:coord>
11    <gml:coord>
12     <gml:X> 4437960.070 </gml:X>
13     <gml:Y> 5331818.450 </gml:Y>
14    </gml:coord>
15    <gml:coord>
16     <gml:X> 4437967.200 </gml:X>
17     <gml:Y> 5331825.410 </gml:Y>
18    </gml:coord>
```

```
19   </gml:LineString></gml:centerLineOf>
20   <attributes>
21    <status> open for traffic </status>
22    <numberOfRoadways denotation="actual number"> 2
23    </numberOfRoadways>
24    <function> road traffic </function>
25    <dedication> communal </dedication>
26   </attributes>
27  </Street>
28 </dlmMember>
29</DLMModel>
```

The assembly of the geometric line pieces proved to be more complicated than we had first expected. While in GML, lines have to be given by a steady stream of coordinates, DLM lines are separated into little pieces and scrambled. There are no hints from the DLM data sets on how and where a line continues, they may even change direction. Worse still, one DLM polygon object may consist of several unconnected polygons and their inside holes are sometimes connected by a hair line to the outside polygon and sometimes they are just unconnectable polygons that happen to be inside another polygon. To set these line pieces in order we wrote an algorithm that puzzles the line pieces together regardless of their given order and direction. It also identifies hairlines as split points and geometrically checks whether a polygon is inside another. If this results in more than one polygon, we split the object into several objects each one carrying a new, derived object ID (see Fig. 2). Other special issues of

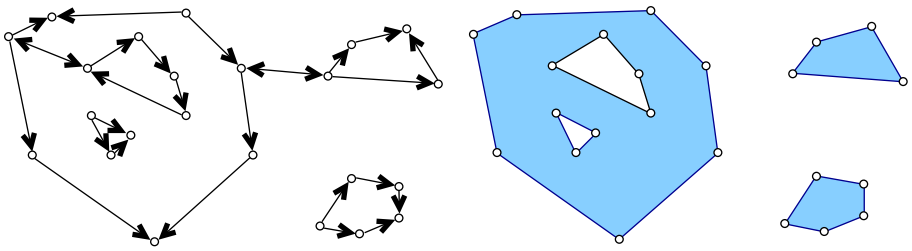

Figure 2. Example of a feasible DLM object (arrows) and three derived GML objects (filled)

the transformation are not covered in detail in this paper. For example, some DLM object types may contain other objects instead of having a geometry. This kind of object references also occurs in context of bridge objects, that we modeled using XLinks to reference the objects under and over the bridge.

3. Scalable Vector Graphics – SVG

For the visualization of the GML documents the XMLbased language of choice was Scalable Vector Graphics (SVG) (Cagle, 2002; Eisenberg, 2002). SVG is especially suitable for our task, as it is a language to describe two-dimensional graphics that can be viewed by an internet browser with an appropriate plug-in (Adobe, 2003). The scalability is also important, as decent maps should be zoom-able and yet manageable in size. SVG has of course much more features than the ones mentioned here, yet for the purpose of map drawing only the static vector part remains of interest.

Path and fill

Most DLM objects are either lines or polygons. So the most important SVG element is ”path”, which allows us to draw lines and to define the outlines of polygons. The instructions themselves have some similarity to plotter instructions. A imaginary pen can be put down, lifted up and be moved. Polygons are closed line-trips and can be filled. To fill a polygon a lot of possibilities are given, from straight forward color to self-defined pattern.

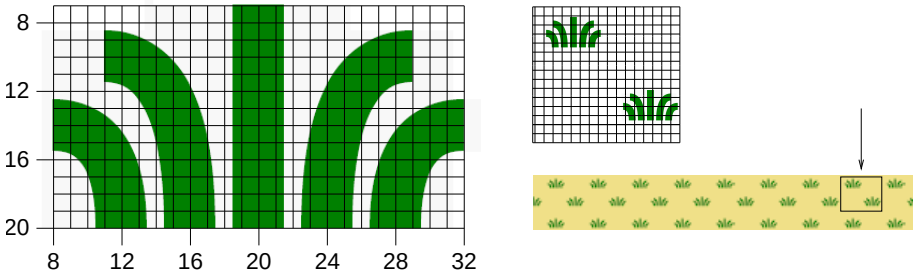

Figure 3. Constructing a pattern and using it

The seven lines below contain a example for the use of the path statement. It is a plant symbol, which is to be used later as a filling pattern for swamp areas (Fig 3). In line 1 some attributes of the path are defined: it is not a filled polygon (fill=”none”), the line has a thickness of 3 pixels (stroke-width=”3”) and the line color is supposed to be green (stroke=”green”). In the following lines 2-6 the attribute ”d” (d: path data) contains the geometric description of the line. This is given by one letter instructions followed by numerical parameters. Most instruction have two versions: capital letters imply absolute coordinates, while lower case letters imply relative coordinates.

In this example, first the ”pen” is moved (M: move to) without drawing to the absolute position (20, 20). From there a straight line (l: line to) is drawn to relative coordinates (0, -13). This draws the middle line of the five plant

symbol lines. The other lines are not straight, so instead of a linear line, a quadratic spline (q: quadratic Bèzier curve) is drawn. The first coordinate is a control point, while the second gives the end of the line. In line 7 the path is given a identification name. This can be used to reference the symbol in a filling pattern. Similar path elements can be used to create other symbols like trees or towers.

```
1 <path fill="none" stroke-width="3" stroke="green"
2   d="M 20 20 l 0 -13
3      M 16 20 q 0 -10 -5 -10
4      M 12 20 q 0  -6 -4  -6
5      M 24 20 q 0 -10  5 -10
6      M 28 20 q 0  -6  4  -6"
7   id="PlantSymbol"/>
```

Filling patterns are defined by the ”pattern” element. An example is shown below. In line 1 the pattern is given an identification name (id=”Swamp”) and a local coordinate system is introduced (line 1+2). The coordinate system is at same scale for all objects, which are therefore drawn at same size all over the map. In the lines 3-7 the content of the pattern is described. First (in line 3) it is filled with the color ”khaki”. Then two of the already known plant symbols are inserted using XLink (lines 4-7). Both are translated to their position in the pattern (compare Fig 3).

```
1 <pattern id="Swamp" x="0" y="0" width="65" height="60"
2   patternUnits="userSpaceOnUse">
3    <rect fill="khaki" x="0" y="0" width="65" height="60"/>
4    <use xlink:href="#PlantSymbol"
5       transform="translate(-2,-2)"/>
6    <use xlink:href="#PlantSymbol"
7       transform="translate(32,30)"/>
8 </pattern>
```

To fill an given area of swamp with the pre-defined pattern, we use the style definitions of the CSS (Cascading Style Sheets (Meyer, 2002)). An excerpt from the style sheet is given below. Line 1 means that an element, in our case a SVG path, that uses the style class ”areaSwamp” is automatically filled with the pattern ”Swamp”, that had been defined in the same document. But style pattern do not have to be pre-defined. In the lines 2-4 we can see an ad hoc style definition made for minor streets.

```
1 .areaSwamp {fill: url(#Swamp)}
2 .lineMinorStreetCommunalForeground
3   {fill: none; stroke-width: 8.5px;
4    stroke: snow; stroke-linejoin: round}
```

The next example shows the final step of drawing a swamp area. Since the style class ”areaSwamp” is known, a path with the class can be assigned. The

path data is automatically extracted from some real GML data and represents a small swamp area.

```
1 <path class="areaSwamp"
2   d="M 1044.21 -2842.24
3      L  993.06 -2794.38  949.65 -2753.24
4         919.2  -2782.51  873.62 -2831.37
5         925.87 -2884.03  961.61 -2923.33
6         986.69 -2899.01 1006.8  -2879.51
7      Z"/>
```

How to produce map-like graphics

To be able to transform all GML-encoded features into SVG, numerous style definitions are needed. We concentrated on the most common feature types (about 50) and defined about 70 style classes and 30 symbols. There are more style classes than feature types, because some feature types are drawn differently depending on the values of their attributes. An example are streets, where highways are drawn differently than minor streets, yet both belong to the same feature type. The other ”trick” about streets is that foreground and the background of streets are drawn on different layers of the map. This avoids the need of complicated calculations on how to draw intersection, as all streets are first drawn as black lines and then overdrawn by slightly thinner colored lines.

To get an appealing map design, we tried to create style classes as close to the original map designs as possible. We also took some inspirations from the cartographers as we organized our maps into layers as they do it (also see (Belussi et al., 2003)). That way unimportant features can be overdrawn without first checking, if they are really less important than the ones we are drawing later. Fortunately, SVG supports this approach by drawing the graphical objects in the order of their appearance in the document. The graphical objects are organized into 11 layers, from ”always overdraw” to ”always on top”. Most of those layers refer to different importance of streets and bridges. As a result we have managed to produce visualization which share close resemblance with the official maps from the office of geographical survey.

4. Transformation of GML features into SVG elements using XSLT

A tool for the transformation of XML documents is the XML language XSLT (extensible style sheet language for transformations). It resembles a functional programming language and transforms XML documents into other XML documents of a different structure. In our context we use it to transform GML into SVG. XML documents – and hence GML documents – have a tree-like structure that is queried by the XSLT for certain patterns. The patterns are given in XSLT templates and if matched, the templates also give instruc-

tions on how to build a piece of the target tree. In our case the result is a SVG document.

In the following sections some XSLT elements are discussed that are important for our task. We present two examples on street objects, how they are transformed from GML into SVG. Other transformations work analog.

Templates for the transformation of streets

Each subtype of DLMMemberType – thus every DLM object that is represented in GML – is transformed by at least one specialized template. The template searches in a given GML document for the appropriate node type. Each of those templates has the attribute ”match”, which contains an XPath expression matching the given node type (Benedikt et al., 2003; Gottlob et al., 2003). In the example template below in line 2 the matching term relates to all streets in the GML document. The streets are further differentiated whether they are communal roads or miscellaneous (line 3+4). If they are, the template ”DrawPath” is called (line 5) with the appropriate parameter (styleclass = ”lineMinorStreetCommunalForeground” in the lines 6+7). DrawPath is called on every node that matches the previous conditions and draws a line with the given style class.

```
1 <xsl:template
2  match="/dlm:Model/dlm:member/dlm:Street
3   [contains(dlm:attributes/dlm:dedication,'communal')
4    or contains(dlm:attributes/dlm:dedication,'misc.')]">
5  <xsl:call-template name="DrawPath">
6   <xsl:with-param name="styleclass"
7      select="'lineMinorStreetCommunalForeground'"/>
8  </xsl:call-template>
9 </xsl:template>
```

There is a template for each different graphical object, e.g. one for communal streets, one for highways, etc. Depending on the geometry different templates are called, e.g. ”DrawPoint” for point objects, each with the styleclass that is used to draw the correct graphical object.

DrawPath: a named template for drawing lines

XSLT has two kinds of templates: template rules as the one seen above and named templates that work very similar to subroutines. While the template rules are directly applied on the source tree, the named templates are invoked by using the ”xsl:call-template” element. The named template may also have parameters which can be set in the xsl:call-template element using ”xsl:with-param” as seen in the above example. We used named templates to automate the repeating task of building SVG path elements like the ones introduced in

section 3. Below is the shortened version of the named template ”DrawPath”, error handling and minor details are omitted to improve readability. ”DrawPath” is used to automatically draw line object such as rivers or streets.

The lines 1 and 2 serve as ”function header” for the named template, defining its name and parameters. The element in line 3 is not part of the XSL namespace and therefore is not an instruction, but an element that is added to the target SVG document. The target ”svg:path” element only contains attributes and no elements, so in the lines 4 and 7 its attributes are defined. The value of the first attribute ”class” can be retrieved by evaluating the parameter ”styleclass” (line 5), which has been defined by the calling template. The style class determines the color, thickness and shape of the line to be drawn.

The data attribute ”d” is more complicated to construct, because its content has to be retrieved from various nodes of the GML document. The ”xsl:for-each” element (lines 8-26) retrieves all coordinate nodes via the XPath expression given by the ”select” attribute. The contents of the nodes are applied to the named templates getX and getY, which retrieve and convert the DLM coordinates into absolute SVG coordinates. The ”M” before the first coordinate pair makes the virtual ”pen” move to the first point without drawing and the ”L” draws lines to all coordinate pairs to come. Since both are capitalized, the coordinates are all interpreted to be absolute. Further coordinates are simply appended, only separated by a space (line 19). Only the last one needs no space (lines 21-23).

```
 1 <xsl:template name="DrawPath">
 2  <xsl:param name="styleclass"/>
 3  <svg:path>
 4   <xsl:attribute name="class">
 5    <xsl:value-of select="$styleclass"/>
 6   </xsl:attribute>
 7   <xsl:attribute name="d">
 8    <xsl:for-each select="gml:centerLineOf/gml:coord">
 9     <xsl:choose>
10      <xsl:when test="position() = 1">
11       <xsl:text>M </xsl:text>
12       <xsl:call-template name="getX"/>
13       <xsl:text> </xsl:text>
14       <xsl:call-template name="getY"/>
15       <xsl:text> L </xsl:text>
16      </xsl:when>
17      <xsl:otherwise>
18       <xsl:call-template name="getX"/>
19       <xsl:text> </xsl:text>
20       <xsl:call-template name="getY"/>
21       <xsl:if test="position() != last()">
22        <xsl:text> </xsl:text>
23       </xsl:if>
```

```
24      </xsl:otherwise>
25     </xsl:choose>
26    </xsl:for-each>
27   </xsl:attribute>
28  </svg:path>
29 </xsl:template>
```

Assuming that the template for communal roads would be applied to the example Bachstr. from section 1, the DrawPath would generate the following SVG path element.

```
1 <path class="lineMinorStreetCommunalForeground"
2   d="M 8245.97 -2142.98
3      L 8253.65 -2146.15
4        8259.83 -2151.12"/>
```

5. The transformation process in summary

In sections 2, 3 and 4 we have described how we use the XML languages GML, SVG and XSLT to transform DLM legacy data into map-like web-visualizations.

In summary: We downloaded the DLM legacy data from the ATKIS-project in the EDBS format. Then we used ”EDBS_Extra” (Rinner, 2001) to transform EDBS into a more readable form. Next we used our JAVA application to transform this data into GML. The open source XML Schema parser ”Xerces” (Apache, 2002) is used to validate the resulting GML against the GML schema we have created to fit the DLM data. The resulting GML file can then be transformed, for example into SVG with the XSLT document that has been specifically designed for that task by application of an XSLT processor.

We used the open source processor ”Xalan” (Apache, 2003) in our test runs, which worked out very well. The XSLT document not only contains transformation templates that define how GML elements are transformed into SVG elements. It also includes the cascading style sheets (CSS) that define how geographical objects are drawn on the map and the predefined pattern used by CSS to create the more complex polygon fills that are used to visualize forests and swamps and the symbols on the map for towers and sights. The resulting SVG bears close resemblance to the original maps including a legend for the available style classes.

Web browsers can be used to watch SVG files on the screen, but most need an installed SVG viewer plug-in. Figure 4 shows a part of the transformed and visualized test data obtained from the Bavarian office for geographical survey. The whole map is 50 centimeters in square and has a scale of 1:25,000. The SVG files are especially suited for web-visualization, due to their small file size im comparison to raster format graphics of a corresponding detail level.

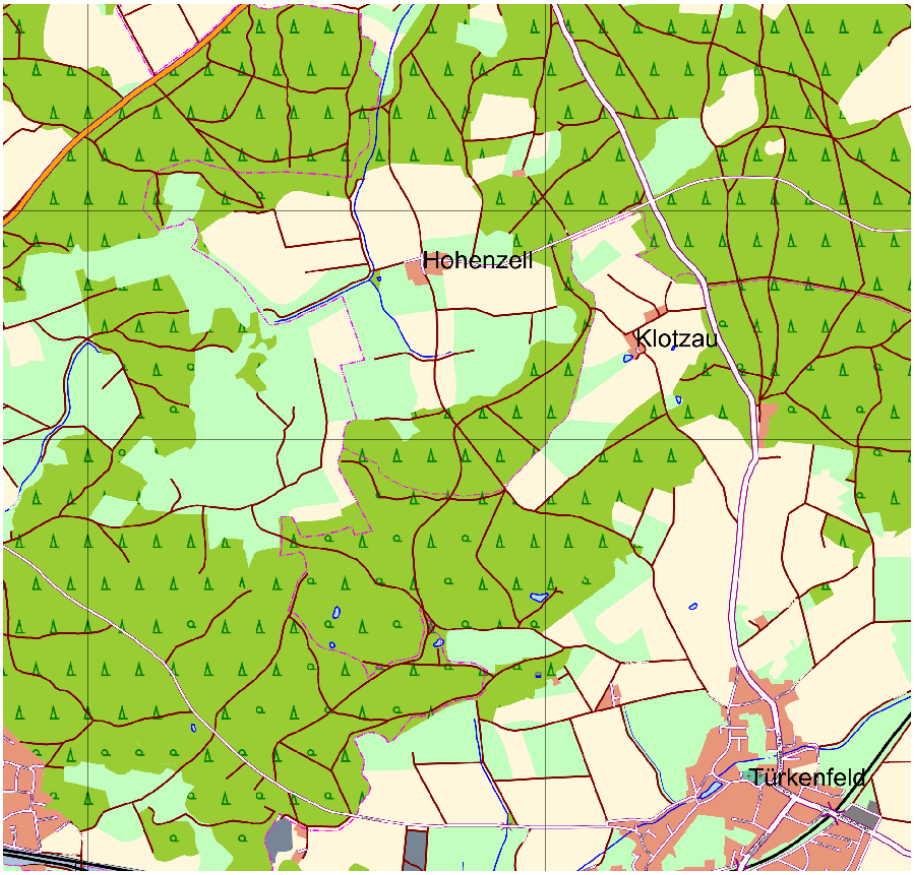

Figure 4. Part of a generated map

All in all we have so far transformed and visualized the data for six different German regions including cities and rural areas with quite pleasing results.

6. Discussion

In our case study we show, that XML languages like GML, SVG and XSLT are well suited to model and visualize the DLM legacy data. The visualization has been tested on realistic data and so far has proven to produce viable maps. The GML modeling has been an important step to bring the legacy data into an easy usable shape. Open source XML tools like ”Xerces” or ”Xalan” has been a great help. Conclusively we have demonstrated that the whole process of

transforming the DLM data into a graphical representation can be effectively accomplished by the exclusive use of XML and JAVA.

We have put no emphasis on running time, trading it for easy implementation and extensibility as the transformations themselves are likely to be used rarely and the running time never exceeded several minutes per map.

The quality of our maps could yet be improved by increasing the number of implemented templates, style classes and symbols. As already mentioned only the most common graphical objects have been implemented so far. Still the most challenging difference between our maps and official maps are adjustments to be made depending on the respective context. For example tree symbols that are close to a street may be partially disguised by that street in our maps. The reason is that trees as well as streets are not drawn according to scale but much larger. So the trees have to be moved to be by the street, which is quite complicated, as the drawn width of streets varies according to their importance and the trees must be moved in a way that does not disguise any other important feature, like another tree or another street. This operation of cartographic displacement could, in principle, also be implemented in XSLT, but a conventional imperative programming language would be better suited.

The GML model, as well as the XSLT transformation can be generalized to fit more than just German legacy data. The overall procedure as outlined in this paper is applicable to all geographical data formats, with only little adjustments. Once the data is transformed to GML, it can be used as a reliable platform for other applications. One of these applications can be a visualization similar to the one we have presented in this paper.

References

Adobe (2003). Adobe Plug-in for SVG. Adobe SVG Zone, http://www.adobe.com/svg/.

Apache (2002). Xerces-Java-Parser, Version 2.3.0. Apache Software Foundation, http://xml.apache.org/dist/xerxes-j/.

Apache (2003). Xalan-Java Version 2.5.1. Apache Software Foundation, http://xml.apache.org/xalan-j/.

Belussi, A., Catania, B., and Bertino, E. (2003). A reference framework for integrating multiple representations of geographical maps. In *Proceedings of the 11th ACM Int. Symp. on Advances in Geographic Information Systems*.

Benedikt, M., Fan, W., and Kuper, G. (2003). Structural properties of xpath fragments. In *Proc. ICDT'03*.

Bex, G.J.., Maneth, S., and Neven, F. (2002). A Formal Model for an Expressive Fragment of XSLT. *Information Systems*, 27(1):21–39.

Cagle, K. (2002). *SVG Programming*. Apress.

E. van der Vlist (2003). *XML Schema*. O'Reilly.

Eisenberg, J.D.. (2002). *SVG Essentials*. O'Reilly.

Fidalgo, R., da Silva, J., Times, V. C., de Souza, F., and de Barros, R. S. (2003). GMLA: A XML Schema for Integration and Exchange of Multidimensional-Geographical data. In *GEOINFO 2003*.

Goldfarb, C.F.. and Prescod, P. (2003). *XML Handbook*. Prentice Hall.

Gottlob, G., Koch, C., and Pichler, R. (2003). Xpath processing in a nutshell. *SIGMOD Rec.*, 32(2):21–27.

Guo, Z., Zhou, S., Xu, Z., and Zhou, A. (2003). G2ST: A Novel Method to Transform GML to SVG. In *11th ACM Int. Symp. on Advances in Geographic Information Systems*, pages 161–168.

Harold, E.R.. and Means, W.S.. (2002). *XML in a Nutshell*. O'Reilly.

Kay, M. (2001). *XSLT Programmer's Reference*. Wrox Press, 2. edition.

Kraak, M.J.. and Brown, A. (2000). *Web Cartography: Developments and Prospects*. Taylor & Francis.

Kraak, M.J.. and Ormeling, F. (2002). *Cartography – Visualization of Spatial Data*. Pearson Higher Education, 2. edition.

Meyer, E.A.. (2002). *On CSS. Mastering the Language of Web Design with Cascading Style Sheets*. New Riders.

OGC (2001). Geography Markup Language (GML) 2.0.0. Open Geospatial Consortium (OGC), http://schemas.opengis.net/gml/2.0.0/.

OGC (2004). Geography Markup Language (GML) 3.1.0. Open Geospatial Consortium (OGC), http://schemas.opengis.net/gml/3.1.0/base/.

Rinner, C. (2001). ATKIS-Reader EDBS_extra. http://www.rinners.de/edbs/.

W3C (1999). XSL Transformations (XSLT), Version 1.0. World Wide Web Consortium (W3C), http://www.w3.org/TR/1999/REC-xslt-19991116.

W3C (2000). Scalable Vector Graphics (SVG) 1.0 Specification. World Wide Web Consortium (W3C), http://www.w3.org/TR/2000/WD-SVG-20000629/.