

# Modellierung und kartographische Visualisierung von Geodaten mit XML-basierten Sprachen

Karl Neumann, Brigitte Mathiak, Andreas Kupfer

TU Braunschweig, Institut für Software, Abteilung Informationssysteme  
www: <http://www.cs.tu-bs.de/idb/> – E-Mail: [K.Neumann@tu-bs.de](mailto:K.Neumann@tu-bs.de)

**Abstract:** Mit den XML-basierten Sprachen GML, XSLT und SVG lassen sich Geodaten nicht nur anwendungsorientiert modellieren sondern auch kartenähnlich visualisieren. In dieser Fallstudie zeigen wir das, indem wir realistische Geodatenbestände der Landesvermessungsämter zunächst mit der Geography Markup Language (GML) nachmodellieren. So mit GML strukturierte Daten werden dann mit der Extensible Stylesheet Language for Transformation (XSLT) auf Elemente der Sprache Scalable Vector Graphics (SVG) abgebildet. Dabei wird der Prozess der kartographischen Visualisierung durch XSLT-Konstrukte modelliert und auch gleichzeitig implementiert. Als Ergebnis erhalten wir Graphiken, die den entsprechenden Karten der Landesämter zumindest nicht unähnlich sind.

## 1 Einleitung

Beim Austausch von Geodaten und bei der rechnerbasierten Kartennutzung wird in letzter Zeit – wie bei zahlreichen anderen Anwendungen – verstärkt das Internet genutzt [HA00, KB00, Te01]. Damit werden auch in diesem Kontext Techniken und spezielle Sprachen immer interessanter, die auf der Trägersprache XML aufbauen. Eine solche Sprache ist die XML-Anwendung GML (Geography Markup Language). Sie wurde vom Open GIS Consortium, einem internationalen Zusammenschluss von ca. 200 Firmen, Behörden sowie Forschungseinrichtungen, als offener Standard zum Austausch von Geodaten konzipiert [OGC01]. Dieser Standard legt mit XML-Sprachmitteln im Wesentlichen ein Geometrieschema und ein Featureschema fest, in deren Rahmen eigene Geoanwendungswelten modelliert werden können. Mit einer weiteren XML-basierten Sprache, der Extensible Stylesheet Language for Transformation (XSLT) lassen sich nun alle Arten von XML-Dokumenten – und damit auch GML-Dokumente – in anders strukturierte XML-Dokumente überführen. Es liegt daher der Gedanke nahe, Geoobjekte mittels XSLT auf graphische Objekte so abzubilden, wie man es aus kartographischen Anwendungen gewohnt ist. Als Zielsprache bietet sich dazu, wie auch in [BW01] und [GZXZ03] vorgeschlagen, wiederum eine XML-basierte Sprache an, Scalable Vector Graphics (SVG). Der kartographische Abbildungsprozess, der mit GML kodierte Geoobjekte auf SVG-Zeichenbefehle abbildet, wird dabei mit XSLT-Konstrukten modelliert und – praktisch automatisch – durch einen XSLT-Prozessor implementiert.

Im vorliegenden Text zeigen wir, wie man konkret die genannten XML-Sprachen einsetzen kann, um Geoinformationen geeignet zu modellieren, und so strukturierte Daten dann zu transformieren, um als Ergebnis kartenähnliche Präsentationsgraphiken zu erhalten. Dazu wird im nächsten Abschnitt zunächst die Sprache GML kurz erläutert, und dann wird die Erarbeitung eines GML-Schemas für Objekte des Digitalen Landschaftsmodells der Vermessungsämter in Ausschnitten und anhand konkreter Beispiele diskutiert. Die für uns wichtigsten Elemente der Graphiksprache SVG werden anschließend in Abschnitt 3 vorgestellt, und die Transformationssprache XSLT sowie die Modellierung des Abbildungsprozesses werden im 4. Abschnitt behandelt. Den Ablauf der gesamten Fallstudie, ausgehend von Testdaten im Format des "Amtlichen Topographisch-Kartographischen Informationssystems" der Landesämter über die umgesetzten GML-Daten bis zu den schließlich generierten Vektorgraphiken, erläutern wir im 5. Abschnitt. Hier werden auch Ausschnitte aus fertigen kartenähnlichen Abbildungen präsentiert. In Abschnitt 6 fassen wir die Ergebnisse unserer Arbeit kurz zusammen und geben einen Ausblick auf weitere mögliche Aktivitäten.

## **2 Modellierung von Geodaten mit der Geography Markup Language**

Die Geography Markup Language (GML) ist eine spezialisierte Auszeichnungssprache, die durch XML sowie XML-Schema erzeugt wird. Die Funktionsweise und die Konstrukte der Metaauszeichnungssprache XML und auch die erweiterten Strukturierungsmöglichkeiten von XML-Schema sollen hier nicht erläutert werden; Informationen dazu finden sich z.B. in [Ra01, HKS02, HM02, Je02, EE04]. Im Mittelpunkt der Modellierungskonstrukte von GML stehen geographische Features, das sind Beschreibungen von Geoobjekten in einem Bezugssystem. Die geometrischen Eigenschaften der Objekte beruhen auf einfachen, höchstens zweidimensionalen Koordinaten, Strecken zwischen zwei Stützpunkten werden stets als Gerade angenommen. Zur besseren Strukturierung wurden drei GML-Teilschemata definiert: das Geometrie-Schema (`geometry.xsd`), das Feature-Schema (`feature.xsd`) und ein Schema, das die Vernetzung mit externen Quellen unterstützt (`xlinks.xsd`). Diese Schemata wurden vom Open GIS Consortium im XML-Format zur freien Verfügung gestellt [OGC01].

### **2.1 Features und Geometrien in GML**

Da GML als offener Standard für möglichst zahlreiche verschiedene Geowanwendungswelten gedacht ist, werden keine Klassen konkreter Geoobjekte definiert, sondern es wird mit dem Begriff der Features eher ein Strukturierungsrahmen zur Modellierung solcher Anwendungswelten vorgegeben. Die Features werden dargestellt durch eine Anzahl von Eigenschaften ("Properties"), die ihrerseits aus Name, Typ und Wert bestehen. Dabei ist die Anzahl der Eigenschaften, sowie deren Name und Typ für jedes Feature durch seinen Featuretyp festgelegt. Ein geographisches Feature hat mindestens eine Eigenschaft, die geographische Daten enthalten kann. Eine Sammlung von Features wird ebenfalls als Fea-

ture betrachtet, dessen Eigenschaften sind dann vom Typ Feature. Zusätzlich dazu kann es noch weitere Eigenschaften geben, die eine Feature-Sammlung feiner beschreiben. Die Struktur von Features, die Eigenschaften enthalten, die eventuell wieder Features enthalten, usw. findet sich in der Struktur der vorgegebenen GML-Oberklasse "AbstractFeatureType" wieder. Es lassen sich damit gut komplexe, geschachtelte Objekte modellieren, etwa Länder, die Städte enthalten, die Straßen enthalten an denen Gebäude stehen.

Die in GML möglichen fünf geometrischen Datentypen werden von einer entsprechenden Oberklasse ("AbstractGeometryType") abgeleitet. Es gibt die Datentypen Punkt (point), Linie (line), Rechteck (box), geschlossener Linienzug (linear ring) und Polygon (polygon). Da ein Linienzug eine geschlossene Fläche beschreiben soll, muss er wenigstens drei unterschiedliche Punkte enthalten, und da außerdem der letzte Punkt gleich dem ersten Punkt sein soll, müssen mindestens 4 Punkte angegeben werden. Anders als bei einem Linienzug sind bei einem Polygon auch beliebig viele Aussparungen (Löcher) möglich. Daher besteht ein GML-Polygon aus einem Linienzug, der die äußere Begrenzung darstellt und aus beliebig vielen weiteren Linienzügen, den Aussparungen. Die folgenden 23 Zeilen zeigen zur Illustration die vorgegebene Definition des Datentyps "PolygonType" aus dem GML-Geometrieschema: Der Datentyp wird als "complexType" definiert (Zeile 1), vom Typ "AbstractGeometryType" abgeleitet (Zeile 3) und enthält als Elemente die äußere Fläche (Zeilen 5–11) und die inneren Flächen (Zeilen 12–19).

```
1 <complexType name="PolygonType">
2   <complexContent>
3     <extension base="gml:AbstractGeometryType">
4       <sequence>
5         <element name="outerBoundaryIs">
6           <complexType>
7             <sequence>
8               <element ref="gml:LinearRing"/>
9             </sequence>
10          </complexType>
11        </element>
12        <element name="innerBoundaryIs"
13              minOccurs="0" maxOccurs="unbounded">
14          <complexType>
15            <sequence>
16              <element ref="gml:LinearRing"/>
17            </sequence>
18          </complexType>
19        </element>
20      </sequence>
21    </extension>
22  </complexContent>
23 </complexType>
```

## 2.2 Eine GML-Modellierung des Digitalen Landschaftsmodells

Mit den durch die GML-Schemata bereitgestellten Sprachmitteln können eigene Anwendungswelten dargestellt werden, d.h. vor allem, neue Featuretypen und deren Beziehungen untereinander definiert werden [NE03]. Wir betrachten als Anwendungswelt das Digitale Landschaftsmodell (DLM) des "Amtlichen Topographisch-Kartographischen Informationssystems" der Landesvermessungsämter (ATKIS-Projekt). Das Projekt verfolgt seit ca. 15 Jahren das Ziel, die gesamte Oberfläche der Bundesrepublik Deutschland digital zu erfassen und zwar in topographischer als auch in kartographischer Hinsicht. Im Teilprojekt "Digitales Landschaftsmodell" wurde u.a. ein verbindlicher Objektartenkatalog herausgegeben, der festlegt, wie einzelne Objekte der Landschaft gebildet werden und welcher von ca. 200 vorgegebenen Objektarten mit welchen Attributen sie zugeordnet werden. So gibt es etwa die Objektarten Wohnbauflächen, Straßen, Seen mit Attributen wie Zustand, Fahrbahnbreite etc. Der aktuelle Stand des Projekts wird in [ADV03] dokumentiert, eine knappe Übersicht enthält z.B. [NE00].

Zur Nachmodellierung des Digitalen Landschaftsmodells mit GML bietet es sich an, als Entsprechung der DLM-Objektarten einen komplexen Typ einzuführen, den wir "AtkisMemberType" nennen und der als Exemplare dann "AtkisMember" enthalten kann. Wie vom GML-Rahmen vorgegeben, wird "AtkisMemberType" als Featuretyp definiert, also vom GML-Typ "AbstractFeatureType" abgeleitet. Alle Objekte der Klasse "AtkisMemberType" haben einen Objektidentifikator ("AtkisOID") und u.a. eine Geometrie sowie evtl. zahlreiche Attribute, wie z.B. "Art der Grenze", "Funktion" oder "Zustand". Als Geometrie-Elemente passen hier gut die von GML vorgegebenen: "location" für Punkte, "centerLineOf" für Linien und "polygonMember" für Polygone. Die drei Namen sind in GML definierte Synonyme für die oben erläuterten Datentypen Punkt, Linie, Polygon. Die folgenden 15 Zeilen zeigen eine vereinfachte Version der Deklaration unserer Klasse "AtkisMemberType": Es wird spezifiziert, dass es sich um einen komplexen Typ handelt (Zeile 1), und in Zeile 3 wird der GML-Typ "AbstractFeatureType" (Präfix "gml") als Supertyp angegeben. Der Objektidentifikator sowie die Attribute werden als Elemente deklariert (Zeilen 5 und 11) und die Geometrie als Auswahl (Zeilen 6–10).

```
1 <complexType name="AtkisMemberType">
2   <complexContent>
3     <extension base="gml:AbstractFeatureType">
4       <sequence>
5         <element ref="AtkisOID" minOccurs="0"/>
6         <choice minOccurs="0">
7           <element ref="gml:location"/>
8           <element ref="gml:centerLineOf"/>
9           <element ref="gml:polygonMember"/>
10        </choice>
11        <element ref="Attribute" minOccurs="0"/>
12      </sequence>
13    </extension>
14  </complexContent>
15 </complexType>
```

Auf ähnliche Weise werden die weiteren Klassen deklariert. So sind die einzelnen Objektarten des Digitalen Landschaftsmodell wie “Straße” oder “Weg” in unserer Modellierung Subtypen des “AtkisMemberType”. Dieser Struktur folgend können nun Objektexemplare angegeben werden, z.B. konkrete Straßen, Grenzen oder Verwaltungsbezirke mit ihren jeweiligen Koordinaten. Die folgenden 25 Zeilen enthalten ein Beispiel dafür: die Angabe einer (sehr kurzen) Straße namens “Badstraße” aus einem umgesetzten DLM-Testdatenausschnitt.

```
1 <AtkisMember>
2 <Strasse>
3 <gml:name> Badstrasse </gml:name>
4 <AtkisOID> 86118065 </AtkisOID>
5 <gml:centerLineOf>
6 <gml:coord> <gml:X> 4437952.980 </gml:X>
7 <gml:Y> 5331812.550 </gml:Y> </gml:coord>
8 <gml:coord> <gml:X> 4437960.070 </gml:X>
9 <gml:Y> 5331818.450 </gml:Y> </gml:coord>
10 <gml:coord> <gml:X> 4437967.200 </gml:X>
11 <gml:Y> 5331825.410 </gml:Y> </gml:coord>
12 </gml:centerLineOf>
13 <Attribute>
14 <Zustand> in Betrieb </Zustand>
15 <AnzahlDerFahrstreifen
16 Bedeutung="tatsaechliche Anzahl"> 2
17 </AnzahlDerFahrstreifen>
18 <Funktion> Strassenverkehr </Funktion>
19 <VerkehrsbedeutungInneroertlich>
20 Anliegerverkehr
21 </VerkehrsbedeutungInneroertlich>
22 <Widmung> Gemeindestrasse </Widmung>
23 </Attribute>
24 </Strasse>
25 </AtkisMember>
```

Der Objektname ist zwar nicht ein Bestandteil der Deklaration des “AtkisMemberType” weiter oben, aber er ist eine Komponente der übergeordneten GML-Klasse “AbstractFeatureType”. Die Zeilen 5–12 enthalten die Liniengeometrie der betrachteten Straße, die auch im Original tatsächlich nur drei Stützpunkte hat. Danach folgen die Straßenattribute mit den konkreten Werten (Zeilen 13–23), wie “Zustand: in Betrieb” oder “Widmung: Gemeindestrasse”. Auf diese Weise können nun alle Objekte aller Objektarten eines DLM-Datenauszugs als “AtkisMember” innerhalb eines übergeordneten “AtkisModells” aufgeführt werden. Beispielsweise enthält einer der von uns umgesetzten Testdatenausschnitte insgesamt 5370 solche Objekte vom Typ “AtkisMember”.

Auf die Darstellung einiger Besonderheiten wurde hier aus Platzgründen allerdings nicht eingegangen. So haben wir etwa die Modellierung von Brücken weggelassen; sie wurden mit Referenzen vom Typ “xlink” realisiert.

### 3 Elemente der Sprache SVG

Zur kartenähnlichen Visualisierung GML-kodierter Daten bietet sich eine weitere XML-basierte Sprache an: SVG (Scalable Vector Graphics). SVG beschreibt zweidimensionale, verschiedenartige Grafiken in XML-Terminologie (vgl. z.B. [W3C00, Ca02, Ei02]), wobei für uns hier nur die statischen Vektorgraphiken von Interesse sind.

#### 3.1 Der Path-Befehl und Füllmuster

Eine der wichtigsten SVG-Anweisungen ist in diesem Zusammenhang der Befehl “path”, mit dem jede Art von Linie oder Polygon erzeugt werden kann. Die zu zeichnende Linie wird dabei mit Konstrukten beschrieben, die starke Ähnlichkeiten mit einer Plottersteuerung haben: Es existiert ein imaginärer Stift, der abgesetzt, angehoben und bewegt werden kann. Polygone werden zunächst wie Linien behandelt und anschließend gefüllt, wobei auch selbstdefinierte Füllmuster verwendet werden können.

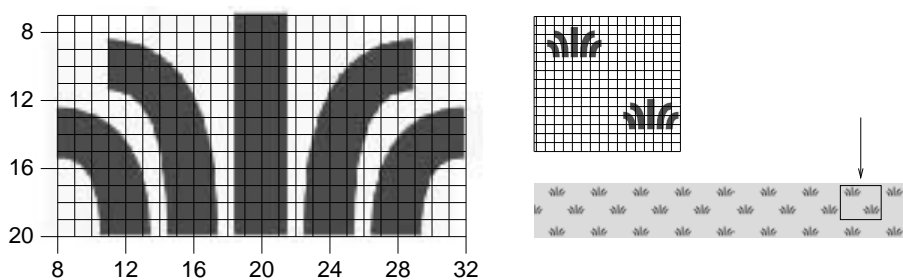


Abbildung 1: Konstruktion einer Signatur und ihre Benutzung als Füllmuster

Die unten aufgeführten 5 Zeilen enthalten ein schon etwas anspruchsvolles Beispiel für die Verwendung des Path-Befehls: Es wird eine Pflanzen-Signatur definiert, die später als Füllmuster für Moor- und Moosvegetationsflächen verwendet wird (vgl. auch Abbildung 1). Zunächst (Zeile 1) wird festgelegt, dass nur Striche und kein gefülltes Polygon gezeichnet werden sollen (`fill="none"`), außerdem werden Strichdicke und -farbe spezifiziert. Danach folgt der eigentliche Zeichenbefehl (Attribut “`d`” in den Zeilen 2–4): Der “Zeichenstift” wird zur Position (20, 20) bewegt (`M`: move to), und von dort wird eine Linie (`l`: line to) mit den zum Ausgangspunkt relativen Koordinaten (0, -13) gezeichnet. Allgemein bedeuten Angaben nach groß geschriebenen Anweisungsabkürzungen (z.B. “`M`” oder “`L`”) absolute Koordinaten und nach klein geschriebenen Anweisungsabkürzungen (z.B. “`l`” oder “`q`”) relative Koordinaten. Im Beispiel folgen noch 4 quadratische Splines (Anweisung “`q`”), die ausgehend von den vier Positionen (16, 20), (12, 20), (24, 20) und (28, 20) gezeichnet werden (Zeilen 3, 4). In Zeile 5 wird dem Befehl – und damit dieser Signatur – ein identifizierender Name gegeben (“Pflanzensymbol”), um die Signatur in weiteren Anweisungen als Ganzes einbinden zu können. Ähnliche Path-Befehle sind

nötig, um z.B. die Symbole für Laub- oder Nadelbäume und etwa auch Signaturen für Denkmäler oder Türme zu erzeugen.

```
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   M 12 20 q 0 -6 -4 -6
4     M 24 20 q 0 -10 5 -10   M 28 20 q 0 -6 4 -6"
5 id="Pflanzensymbol"/>
```

Füllmuster werden mit dem Befehl “pattern” definiert, wie das unten aufgeführte Beispiel zeigt. Das Muster bekommt einen Namen (hier: “Moor”), und es wird ein lokales Koordinatensystem angelegt, das in Abhängigkeit zum Koordinatensystem des aufrufenden Objektes steht und eine Größe für das Füllmuster festlegt (Zeilen 1 und 2). Jenseits dieser Koordinaten wird das Muster automatisch wiederholt, wenn damit eine Fläche ausgefüllt werden soll. Im Beispiel wird das Musterrechteck mit der Farbe “khaki” gefüllt (Zeile 3) und danach werden zwei oben definierte Pflanzensymbole darauf platziert (Zeilen 5–8). Die Pflanzensymbole werden dabei durch einen Zeiger vom Typ “xlink” referenziert. Das Attribut “transform” führt die Verschiebung an die gewünschten Stellen durch.

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

Um eine konkrete Fläche, etwa ein Moorgebiet, nun mit einem bestimmten Muster zu füllen, muss noch der entsprechende Path-Befehl mit dem jeweiligen Muster verbunden werden. Dazu benutzen wir einfache Style-Definitionen (Cascading Style Sheets, [Me02]), z.B. folgende:

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

Diese erste Definition (Zeile 1) bedeutet, dass für die Klasse “gebietMoor” das weiter oben definierte Füllmuster festgelegt wird. Und die zweite Definition (Zeilen 2–4) legt Linienattribute für Nebenstraßen fest. Die gewünschte Style-Definition kann nun als Klassenangabe in einem Path-Befehl benutzt werden, wie das Beispiel in den folgenden vier Zeilen zeigt: Hier wurde ein Zeichenbefehl generiert, der ein (kleines) Moorgebiet zeichnet.

```

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

```

### 3.2 Vorgehensweise zur Erzeugung von kartenähnlichen Graphiken

Damit nun alle Objekte aller Objektarten, die mit GML kodiert als Quellbaum, d.h. als ein konkretes Exemplar vom Typ "AtkisModell", vorliegen können, als SVG-Zeichenbefehle dargestellt werden können, werden zahlreiche solcher Style-Definitionen benötigt. Wir haben uns auf ca. 50 Objektarten konzentriert und dafür ca. 70 Style-Definitionen sowie ca. 30 Signaturen erstellt. Die Anzahl der Style-Definitionen ist deshalb größer als die Anzahl der behandelten Objektarten, weil manche Objektarten zwei Style-Definitionen benötigen. So werden z.B. viele Straßen durch zwei verschieden dicke Striche in zwei verschiedenen Farben dargestellt, die übereinander gezeichnet werden und damit eine dickere Linie mit dünnen Rändern ergeben. Mit den bereitgestellten Style-Definitionen und Signaturen können, wie im folgenden Abschnitt skizziert wird, GML-Objekte durch Mustervergleiche innerhalb von XSLT-Templates auf SVG-Zeichenbefehle abgebildet werden. Um insgesamt eine möglichst ansprechende kartenähnliche Präsentationsgraphik zu erhalten, müssen natürlich die Style-Definitionen und die Signaturen sehr sorgfältig erarbeitet werden. Außerdem ist die Reihenfolge wichtig, mit der die fertigen Graphikobjekte letztlich gezeichnet werden, da SVG überdeckend zeichnet: später gezeichnete Objekte überdecken frühere. Wir haben die Graphikobjekte deshalb in 11 Ebenen organisiert, ähnlich zu den in der Kartographie üblichen verschiedenen Druckfolien [HGM02, KO02]. Insgesamt haben wir uns bei der von uns angestrebten kartenähnlichen Visualisierung der GML-Daten von einer entsprechenden topographischen Karte der Landesvermessung und Geobasisinformation Niedersachsen [LGN99] leiten lassen.

## 4 Transformation von GML-Objekten in SVG-Anweisungen mit XSLT

Ein Werkzeug zur Transformation von XML-Dokumenten ist die XML-Sprache XSLT (Extensible Stylesheet Language for Transformation). Sie ist einer funktionalen Programmiersprache ähnlich und überführt XML-Dokumente – und damit auch GML-Dokumente – in XML-Dokumente anderer Struktur [W3C99, Ka01, BMN02]. Wie die Quelldokumente, bei uns sind das mit GML kodierte Daten des Digitalen Landschaftsmodells, sind dabei auch die Zieldokumente Bäume, und auch das jeweilige "XSLT-Programm" (Style-Sheet) hat eine Baumstruktur. Bei der Transformation wird nach dem Prinzip des "Pattern Matching" vorgegangen, der Quellbaum wird also auf das Vorkommen von Mustern hin untersucht. Diese Muster werden in Form von XSLT-Templates vorgegeben. Trifft ein Muster zu, steht ebenfalls im jeweiligen Template, wie ein Stück des Zielbaums konstruiert werden soll. Da ein Quellbaum auch mehrfach durchlaufen werden kann und dabei im Prinzip

beliebige Äste in den Zielbaum eingepasst werden können, sind Quell- und Zielbäume von sehr unterschiedlicher Struktur möglich. Wir diskutieren im Folgenden einige für unsere Anwendung wichtige XSLT-Elemente. Dabei gehen wir beispielhaft auf zwei Templates ein, die Straßenobjekte auf SVG-Zeichenbefehle mit passenden Parametern abbilden. Die Abbildung anderer linien- oder flächenförmiger Objekte geschieht analog.

#### 4.1 Template zur Abbildung von Straßen

Jeder Subtyp von “AtkisMemberType” – also jede von uns in GML repräsentierte Objektart des Digitalen Landschaftsmodells – wird durch mindestens ein spezialisiertes Template umgesetzt. Dieses Template sucht innerhalb eines gegebenen GML-Quelldokuments nach dem jeweils passenden Knotentyp. Dazu wird innerhalb des Templates das Attribut “match” benutzt, nach dem ein XPath-Ausdruck als Argument erlaubt ist. Im Beispiel-Template für Gemeinde- und sonstige Straßen (Zeilen 1–13, unten) ist der XPath-Ausdruck die Zeichenkette “/d1m:AtkisModell/d1m:AtkisMember/d1m:Strasse” (Zeile 3). Wenn ein Straßen-Knoten gefunden wurde, wird überprüft, ob es in der Menge der Attribute einen Widmung-Knoten gibt und ob dieser den Wert “Gemeindestraße” oder “Sonstiges” aufweist (Zeilen 4–7). Falls das zutrifft, wird ein weiteres Template aufgerufen (Zeile 8), das den Namen “DrawPath” hat. Dieses Template, das im nächsten Unterabschnitt beschrieben wird, ist zuständig für das Zeichnen aller Arten von Linien und damit eben auch für linienhafte Straßensignaturen. Das Aussehen der jeweiligen Linie wird mit der passenden Style-Definition (vgl. Abschnitt 3) gesteuert, die dem Template “DrawPath” als Parameter “styleclass” mitgegeben wird. In unserem Beispiel hat der Parameter den Wert “linieNebenstrasseNahverkehrVordergrund” (Zeile 10).

```
1 <xsl:template
2   match=
3     "/d1m:AtkisModell/d1m:AtkisMember/d1m:Strasse">
4   <xsl:if test="contains(d1m:Attribute/
5                 d1m:Widmung, 'Gemeindestrasse')
6                 or contains(d1m:Attribute/
7                 d1m:Widmung, 'Sonstiges')">
8     <xsl:call-template name="DrawPath">
9       <xsl:with-param name="styleclass" select=
10         "'linieNebenstrasseNahverkehrVordergrund'">
11     </xsl:call-template>
12   </xsl:if>
13 </xsl:template>
```

Entsprechend gibt es zahlreiche weitere Templates, die jeweils für eine graphisch darzustellende Objektart zuständig sind, so z.B. Templates für Flüsse, Seen, Wälder oder Wohnbauflächen. Templates für Objekte mit punktförmiger Geometrie rufen dann ein Template auf, das alle Arten von Punktsignaturen zeichnet, und Templates für linienförmige Objekte das im Folgenden behandelte Template.

## 4.2 Template zur Darstellung von Linien

Die Sprache XSLT umfasst neben Aufrufen von benannten oder unbenannten Templates, die Parameter enthalten können, noch weitere Funktionen wie Auswahl, Abfrage usw. Einige Beispiele enthält der folgende Code des erwähnten Templates “DrawPath”, wobei aus Platzgründen allerdings zahlreiche Vereinfachungen vorgenommen wurden, z.B. sind alle Anweisungen zur Fehlerbehandlung weggelassen worden. In diesem zweiten Template wird ein SVG-Befehl “path” konstruiert (Zeilen 3–29), der in Abschnitt 3.1 bereits im Zusammenhang mit einer Mustersignatur erläutert wurde. Hier werden nun alle Arten linienhafter Objekte damit gezeichnet, also u.a. Straßen und Flüsse.

```
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
9         select="gml:centerLineOf/gml:coord">
10        <xsl:choose>
11          <xsl:when test="position() = 1">
12            <xsl:text>M </xsl:text>
13            <xsl:call-template name="getX"/>
14            <xsl:text> </xsl:text>
15            <xsl:call-template name="getY"/>
16            <xsl:text> L </xsl:text>
17          </xsl:when>
18          <xsl:otherwise>
19            <xsl:call-template name="getX"/>
20            <xsl:text> </xsl:text>
21            <xsl:call-template name="getY"/>
22            <xsl:if test="position() != last()">
23              <xsl:text> </xsl:text>
24            </xsl:if>
25          </xsl:otherwise>
26        </xsl:choose>
27      </xsl:for-each>
28    </xsl:attribute>
29  </svg:path>
30 </xsl:template>
```

Zunächst wird der Wert des Attributs “class” des zu generierenden Knotens auf den an dieses Template übergebenen Styleclass-Parameter gesetzt (Zeilen 4, 5). Dieser Parameter steuert – wie bereits erwähnt – das nähere Aussehen der zu generierenden Linie, also Breite, Farbe etc. Die verbleibenden Anweisungen (Zeilen 7–28) dienen dazu, den Wert des Attributs “d” des SVG-Knoten “path” zu berechnen. Dieses Attribut enthält u.a. die Koordinaten der Linie. Mit der Anweisung “for-each” und dem angegebenen XPath-Ausdruck

(Zeilen 8, 9) werden, ausgehend vom aktuellen Objekt, alle X- und Y-Koordinaten nacheinander geholt und in das Attribut “d” geschrieben. Dazu wird vor die erste Koordinate ein “M” gesetzt (Zeilen 11, 12) und vor die zweite ein “L” (Zeile 16). Alle nachfolgenden Koordinaten werden dann nur durch ein Leerzeichen getrennt (Zeilen 18–25), wobei nach der letzten Koordinate kein Leerzeichen mehr benötigt wird (Zeile 22).

Geht man davon aus, dass das weiter oben diskutierte Template (Abschnitt 4.1), das nach Gemeindestraßen sucht, auf den Knoten vom Typ “AtkisMember” aus Abschnitt 2.2 (“Badstraße”) angesetzt wird, so würde dieses Template das Template “DrawPath” aufrufen, das dann die folgenden drei SVG-Kodezeilen generieren würde:

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

Vergleicht man die ursprünglichen Koordinaten des mit GML kodierten DLM-Objekts “Badstraße”, die aus den Testdaten übernommene Gauß-Krüger-Koordinaten sind, mit den Koordinaten des generierten SVG-Befehls, so sieht man, dass eine Koordinatentransformation durchgeführt wurde. Diese Transformation dient der Anpassung an das Koordinatensystem der SVG-Zeichenfläche und wird innerhalb der Templates “getX” und “getY” durchgeführt, die im Template “DrawPath” aufgerufen werden. Da die Templates “getX” und “getY” keine Besonderheiten aufweisen, haben wir sie hier – wie zahlreiche andere Details – nicht weiter diskutiert.

## 5 Der Transformationsprozess insgesamt

Mit den in den Abschnitten 2, 3 und 4 skizzierten Elementen der Sprachen GML, XSLT und SVG haben wir verschiedene Testdatenbestände des Digitalen Landschaftsmodells, die im Netz frei verfügbar sind, nun kartenähnlich visualisiert. Dabei sind wir wie folgt vorgegangen: Zunächst wurden die Testdaten in einem vom ATKIS-Projekt vorgegebenen Format (so genanntes EDBS-Format, [ADV03]) aus dem Internet geladen. Die Testdaten werden in dieser Form in verschiedenem Umfang von vielen Landesvermessungsämtern bereitgestellt. Mit einem frei verfügbaren Programm (“EDBS.EXTRA”, [Ri01]) wurden diese Daten dann in ein Format umgesetzt, das sich leichter weiterverarbeiten lässt: Die umgesetzten Dateien sind ähnlich einer relationalen Datenbank strukturiert. Die Objekte haben einen Objektidentifikator, und die einzelnen Dateien sind tabellenartig aufgebaut, wobei die Identifikatoren als Schlüssel benutzt werden können. Damit können die Informationen einzelner Objekte aus den drei wichtigsten der erzeugten Dateien “Attribute”, “Namen” und “Linien” relativ leicht rekonstruiert werden. Parallel zum Umformatieren der originalen Testdaten wurde das in Abschnitt 2.2 skizzierte GML-Schema für die Objekte des Digitalen Landschaftsmodells erarbeitet und schließlich mit dem frei verfügbaren Parser “Xerces” [ASF02] gegen die GML-Spezifikation validiert. Das Umsetzen der vorformatierten Testdaten in GML-Dokumente, die dem zuvor erarbeiteten Schema folgen, war eine größere Aufgabe. Sie wurde durch die Implementierung eines Java-Programmpaketes

gelöst. Dieses parst die Eingabedateien, wobei auf das Paket “RegEx” zurückgegriffen wird, das den Umgang mit regulären Ausdrücken stark vereinfacht, danach werden die Geometrien der einzelnen Objekte zusammengesetzt und abschließend wird jeweils ein Testdatenauszug als ein GML-Dokument auf eine Zieldatei geschrieben [Ma03].

Diese Zieldateien sind gleichzeitig Quelldateien für die in Abschnitt 4 erläuterte Transformation der GML-Dokumente in SVG-Dokumente. Die eigentliche Transformation wurde mit dem frei verfügbaren XSLT-Prozessor “Xalan” [ASF03] durchgeführt; vorher musste aber natürlich die Methodik dieser Umsetzung erarbeitet und mit XSLT kodiert werden. Außerdem waren noch die in Abschnitt 3 erklärten Style-Definitionen und Signaturen zu implementieren [Ku03].

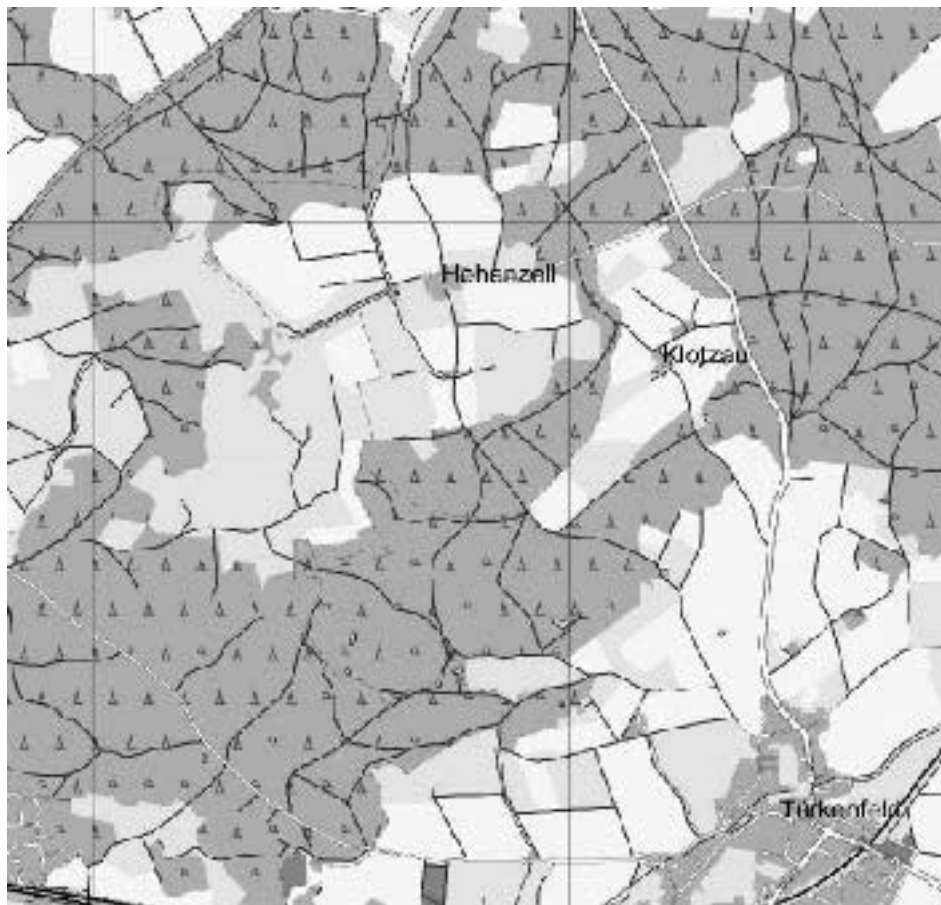


Abbildung 2: generierte kartenähnliche Graphik (Ausschnitt)

Zur graphischen Darstellung fertiger SVG-Dateien auf einem Bildschirm können übliche Web-Browser mit einem SVG-Viewer-Plugin benutzt werden. Abbildung 2 zeigt einen

Ausschnitt der so visualisierten und umgesetzten Testdaten des Gebietes “Türkenfeld” des Bayerischen Landesvermessungsamts. Die gesamte generierte Karte hat die Größe 50 mal 50 cm und ist im Maßstab 1:25.000 angelegt. Neben den Testdaten des Bayerischen Landesvermessungsamts haben wir noch Datenbestände von fünf weiteren Landesämtern umgesetzt und visualisiert.

## 6 Ausblick

In der vorliegenden Fallstudie haben wir versucht, anhand realistischer Datenbestände zu zeigen, dass die XML-basierte Sprache GML gut eingesetzt werden kann, um Geoanwendungswelten zu modellieren, die zugehörigen Daten im entsprechenden Format zu speichern und schließlich auch kartenähnlich zu visualisieren. Dazu haben wir ein GML-Schema für ATKIS-Daten des Digitalen Landschaftsmodells angegeben und Testdaten verschiedener Landesvermessungsämter in das GML-Format überführt. Anschließend wurden diese GML-Daten mit einem XSLT-Prozessor in die XML-basierte Graphiksprache SVG transformiert. Dabei wurde die kartenähnliche Abbildung der Daten durch zwei Arten von Templates, vordefinierte Muster und spezialisierte Style-Definitionen erreicht. Insgesamt sollte demonstriert werden, dass der gesamte Prozess der Strukturierung von Geodaten sowie deren kartographische Darstellung nur mit XML-basierten Sprachen realisiert werden kann, wobei ausschließlich frei verfügbare Software und Werkzeuge eingesetzt wurden. Fragen der Laufzeit und Wartbarkeitsaspekte standen dabei nicht im Vordergrund des Interesses. Auch konnte unser GML-konformes Schema für Objekte des Digitalen Landschaftsmodells nicht mit anderen Schemata verglichen werden, da für diese Anwendungswelt bislang keine alternativen GML-Schemata bekannt sind.

Die Qualität unserer kartenähnlichen Graphiken könnte dadurch gesteigert werden, dass weitere Templates, Style-Definitionen und Mustersignaturen implementiert würden, da wir bislang – wie erwähnt – nur ca. 50 Objektarten berücksichtigt haben. Deutlich schwieriger als das Darstellen zusätzlicher Objektarten ist es dagegen, nebeneinanderliegende Objekte in ihrem kartographischen Kontext zu beachten. So kann es in unseren Darstellungen geschehen, dass etwa die Signaturen einzelner Alleebäume durch eine linienhafte Straßensignatur teilweise verdeckt werden. Die Verdeckung ergibt sich, weil sowohl Bäume als auch Straßen auf Karten nicht maßstäblich sondern viel größer dargestellt werden. Hier müssten also die Koordinaten der Alleebaum-Objekte nicht nur einfach transformiert werden, sondern sie müssten abhängig vom Abstand zur Liniengeometrie der benachbarten Straße vorher von Fall zu Fall etwas verschoben werden. Dabei muss natürlich beachtet werden, dass jetzt nicht die Signaturen anderer wichtiger Objekte verdeckt werden. Diese recht komplizierte Operation der kartographischen Verdrängung [Bu01, Bo02] könnte im Prinzip zwar auch mit XSLT implementiert werden, eine konventionelle imperative Programmiersprache wäre dazu aber besser geeignet.

## Literatur

- [ADV03] ATKIS – Amtliches Topographisch-Kartographisches Informationssystem. Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV), <http://www.atkis.de/>. 2003.
- [ASF02] Xerces-Java-Parser, Version 2.3.0. Apache Software Foundation, <http://xml.apache.org/dist/xerces-j/>. 2002.
- [ASF03] Xalan-Java Version 2.5.1. Apache Software Foundation, <http://xml.apache.org/xalan-j/>. 2003.
- [BMN02] Bex, G.J., Maneth, S., und Neven, F.: A Formal Model for an Expressive Fragment of XSLT. *Information Systems*. 27(1):21–39. 2002.
- [Bo02] Bobrich, J.: Kartographische Verdrängung mit MOVE. *Mitteilungen des Bundesamtes für Kartographie und Geodäsie*. (22):19–29. 2002.
- [Bu01] Burghardt, D.: *Automatisierung der kartographischen Verdrängung mittels Energieminimierung*. Number H.536 in Reihe C. Deutsche Geodätische Kommission. 2001.
- [BW01] Brinkhoff, T. und Weitkämper, J.: Eine Architektur zur XML-basierten Repräsentation von bewegten Geo-Objekten. In: Heuer, A., Leymann, F., und Priebe, D. (Hrsg.), *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW 2001)*. Informatik Aktuell. S. 144–153. 2001.
- [Ca02] Cagle, K.: *SVG Programming*. Apress. 2002.
- [EE04] Eckstein, R. und Eckstein, S.: *XML und Datenmodellierung. XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*. dpunkt.verlag. 2004.
- [Ei02] Eisenberg, J.D.: *SVG Essentials*. O'Reilly. 2002.
- [GZXZ03] Guo, Z., Zhou, S., Xu, Z., und Zhou, A.: G2ST: A Novel Method to Transform GML to SVG. In: *11th ACM Int. Symp. on Advances in Geographic Information Systems*. S. 161–168. 2003.
- [HA00] Herrmann, C. und Asche, H.: *Web.Mapping I: Raumbezogene Information und Kommunikation im Internet*. Wichmann. 2000.
- [HGM02] Hake, G., Grünreich, D., und Meng, L.: *Kartographie – Visualisierung raum-zeitlicher Informationen*. de Gruyter. 8. Aufl. 2002.
- [HKS02] Hansch, M., Kuhlins, S., und Schader, M.: XML-Schema. *Informatik-Spektrum*. 25(5):363–366. 2002.
- [HM02] Harold, E.R. und Means, W.S.: *XML in a Nutshell*. O'Reilly. 2002.
- [Je02] Jeckle, M.: XML-Schemasprachen und W3Cs XML-Schema. In: Glinz, M. und Müller-Luschnat, G. (Hrsg.), *Modellierung 2002*. Band 12 aus *LNI*. S. 23–35. 2002.
- [Ka01] Kay, M.: *XSLT Programmer's Reference*. Wrox Press. 2. Aufl. 2001.
- [KB00] Kraak, M.J. und Brown, A.: *Web Cartography: Developments and Prospects*. Taylor & Francis. 2000.
- [KO02] Kraak, M.J. und Ormeling, F.: *Cartography – Visualization of Spatial Data*. Pearson Higher Education. 2. Aufl. 2002.

- [Ku03] Kupfer, A. Visualisierung von GML-Daten mit XSLT und SVG. Diplomarbeit, TU Braunschweig. 2003.
- [LGN99] Bohnte, Topographische Karte 1:25000, ATKIS. Landesvermessung + Geobasisinformation Niedersachsen (Hrsg.). 1999.
- [Ma03] Mathiak, B. Eine GML-Modellierung für ATKIS-DLM-Daten. Diplomarbeit, TU Braunschweig. 2003.
- [Me02] Meyer, E.A.: *On CSS. Mastering the Language of Web Design with Cascading Style Sheets*. New Riders. 2002.
- [NE00] Neumann, K. und Eckstein, S.: Einführung in die Unified Modeling Language am Beispiel von ATKIS. *Mitteilungen des Bundesamtes für Kartographie und Geodäsie*. (17):81–88. 2000.
- [NE03] Neumann, K. und Eckstein, S.: Geography Markup Language (GML) - Eine Einführung aus Informatik-sicht. *Mitteilungen des Bundesamtes für Kartographie und Geodäsie*. (24):103–111. 2003.
- [OGC01] Geography Markup Language (GML) 2.0. Open GIS Consortium (OGC), <http://www.opengis.net/gml/01-029/GML2.html>. 2001.
- [Ra01] Ray, E.T.: *Einführung in XML*. O'Reilly. 2001.
- [Ri01] Rinner, C. Der ATKIS-Reader EDBS\_extra. <http://www.rinners.de/edbs/>. 2001.
- [Te01] Teege, G.: Geodaten im Internet – Ein Überblick. *Informatik-Spektrum*. 24(4):193–206. 2001.
- [W3C99] XSL Transformations (XSLT), Version 1.0. World Wide Web Consortium (W3C), <http://www.w3.org/TR/1999/REC-xslt-19991116>. 1999.
- [W3C00] Scalable Vector Graphics (SVG) 1.0 Specification. World Wide Web Consortium (W3C), <http://www.w3.org/TR/2000/WD-SVG-20000629/>. 2000.