Der Einsatz von GML, XSLT und SVG am Beispiel von ATKIS-DLM-Daten [1]

(mit 5 Bildern)

Von Karl Neumann, Brigitte Mathiak und Andreas Kupfer, Braunschweig

ZUSAMMENFASSUNG: Mit den XML-basierten Sprachen GML, XSLT und SVG lassen sich Geodaten sowohl strukturieren als auch kartenähnlich visualisieren. In dieser Fallstudie zeigen wir das, indem wir das Digitale Landschaftsmodell von ATKIS mit GML nachmodellieren und diese Daten dann mit Hilfe von XSLT in SVG-Vektorgraphiken umsetzen.

ABSTRACT: Geo data can both be structured and visualized by using the XML based languages GML, XSLT, and SVG. In this case study we first show this by presenting a GML schema for the ATKIS' digital landscape model. Then we demonstrate how these data can be converted to map-like vector graphics by means of an XSLT transformation.

1 Einleitung

Beim Austausch von Geodaten und bei der rechnerbasierten Kartennutzung wird in letzter Zeit – wie bei zahlreichen anderen Anwendungen auch – verstärkt das Internet genutzt. Damit rücken Techniken und spezielle Sprachen, die auf der Trägersprache XML aufbauen, mehr und mehr in der Vordergrund der Interesse. XML (Extensible Markup Language) ist eine Metasprache zur Beschreibung und Erzeugung von Auszeichnungssprachen (vgl. z.B. *Ray* 2001, *Harold/Means* 2002). Und die XML-Anwendung GML (Geography Markup Language) scheint sich als Standard zur Repräsentation und zur Darstellung von Geoinformationen zu entwickeln. GML kodiert, etwas verkürzt ausgedrückt, die Feature- und Geometrie-Schemata des Opengis-Consortiums in XML-Schreibweise (*Open GIS Consortium* 2001).

Mit anderen XML-Anwendungen können nun Geoinformationen nicht nur weltweit per Internet ausgetauscht werden, sondern die Informationen können auch so transformiert werden, dass kartenähnliche Präsentationsgraphiken entstehen, die dann wiederum unkompliziert per Internetbrowser dargestellt werden können. Eine solche XML-Anwendung ist die Sprache XSLT (Extensible Stylesheet Language for Transformation). Mit XSLT können XML-Dokumente in XML-Dokumente anderer Struktur überführt werden. Man kann also z.B. GML-Objekte damit auf geeignete Graphikobjekte abbilden, und für diese Graphikobjekte bietet sich eine weitere XML-Anwendung an, nämlich SVG (Scalable Vector Graphics).

Im vorliegenden Text wird dargestellt, wie man konkret diese XML-Sprachen einsetzen kann, um Geoinformationen geeignet zu strukturieren, und so strukturierte Daten dann

[1] Erscheint in *Mitteilungen des Bundesamtes für Kartographie und Geodäsie, Frankfurt M., 2004.*

auch kartenähnlich präsentieren kann. Dazu verwenden wir ATKIS-DLM-Daten, die von verschiedenen Landesvermessungsämtern im EDBS-Format im Internet als Testdaten allgemein zur Verfügung gestellt werden.

Im nächsten Abschnitt wird zunächst die Sprache GML kurz erläutert, und dann wird die Erarbeitung eines GML-Schemas für Objekte des ATKIS-DLM in Ausschnitten und anhand konkreter Beispiele diskutiert. Die für uns wichtigsten Elemente der Transformationssprache XSLT werden anschließend in Abschnitt 3 durch zwei typische Templates dargestellt, die DLM-Straßenobjekte auf SVG-Zeichenbefehle mit passenden Parametern abbilden. Funktionsprinzipien der Graphik-Sprache SVG und weitere Beispiele konkret benutzter Zeichenbefehle sind dann Inhalt des 4. Abschnitts. Die Transformation von Testdaten im EDBS-Format über die umgesetzten GML-Daten bis zu den schließlich generierten Vektorgraphiken wird als Gesamtprozess in Abschnitt 5 skizziert, bevor wir zum Schluss die Ergebnisse unserer Arbeit kurz zusammenfassen.

Unsere Ausführungen in den Abschnitten 2, 3 und 4 stellen im wesentlichen eine stark komprimierte Sicht auf die Arbeiten von *Kupfer* 2003 und *Mathiak* 2003 dar, auf die wir deshalb an dieser Stelle ausdrücklich verweisen wollen. Bei der von uns angestrebten kartenähnlichen Visualisierung der ATKIS-DLM-Daten haben wir uns von der entsprechenden topographischen Karte der *Landesvermessung + Geobasisinformation Niedersachsen* 1999 leiten lassen.

2 Eine GML-Modellierung für ATKIS-DLM-Daten

GML ist eine spezialisierte Auszeichnungssprache, die durch XML erzeugt wird. Mit dieser spezialisierten Sprache wird ein Beschreibungsrahmen vorgegeben, innerhalb dessen Anwender ihre Geoinformationen strukturieren können. Die so strukturierten Daten können dann problemlos z.B. über die durch das Internet gegebene Technologie mit allen anderen Anwendern ausgetauscht werden, die ebenfalls die Auszeichnungssprache GML benutzen. Wie in *Neumann/Eckstein* 2003 ausgeführt, stellt GML ein Geometrie-Schema (geometry.xsd), ein Feature-Schema (feature.xsd) und ein Schema bereit, das die Vernetzung mit externen Quellen unterstützt (xlinks.xsd). Mit den bereitgestellten Sprachmitteln können eigene Anwendungswelten dargestellt werden, d.h. vor allem, neue Featuretypen und deren Beziehungen untereinander definiert werden.

Betrachtet man als Anwendungswelt das Digitale Landschaftsmodell (DLM) von ATKIS, so bietet sich als zentrale Klasse eine Entsprechung der DLM-Objekte an. Die DLM-Objekte modellieren die Landschaft bekanntlich nach topographischen Gesichtspunkten, und Beispiele für DLM-Objekte sind etwa Grundstücke, Seen oder Straßen. In *Neumann/Eckstein* 2000 wurde eine Modellierung dieser Anwendungswelt u.a. mit UML-Klassendiagrammen angegeben, die als Ausgangspunkt für eine entsprechende Modellierung mit GML genutzt werden kann. Die verschiedenen DLM-Objektarten repräsentieren wir hier als Spezialisierung des übergeordneten Typs “NormalMemberType”, der über mehrere Zwischenklassen u.a. mit der von GML vorgegebenen Klasse “AbstractFeatureCollectionType” in Verbindung steht (vgl. Bild 1, die grau unterlegten Rechtecke sind selbst definierte Klassen, die weißen Rechtecke sind GML-Klassen).

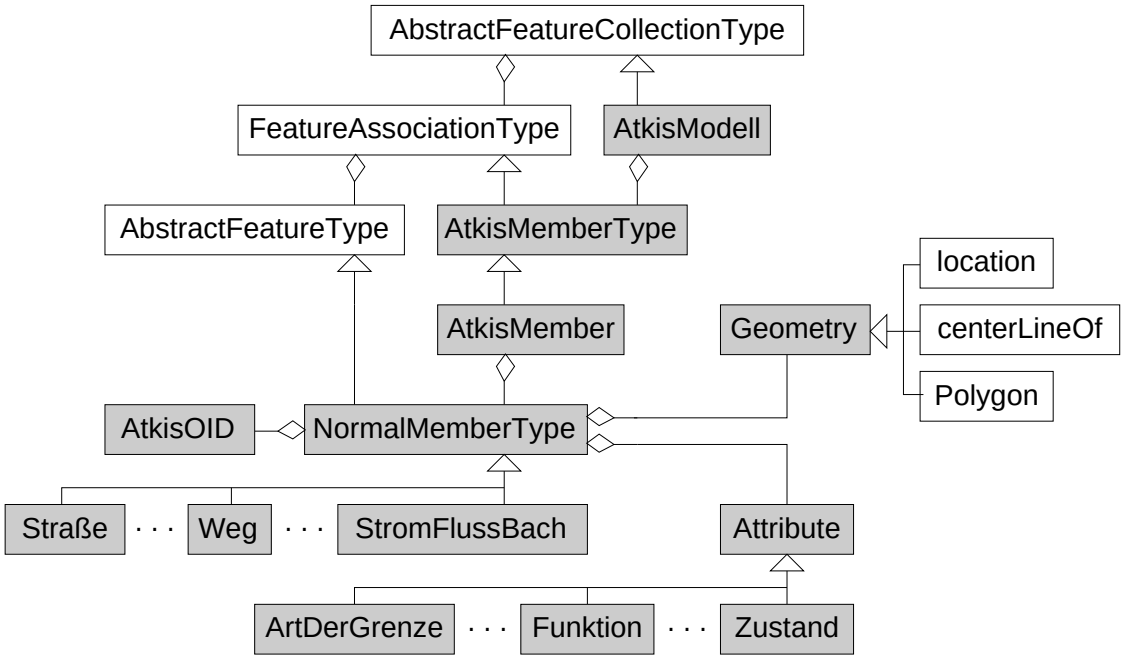

Bild 1 - Ausschnitt der ATKIS-DLM-GML-Modellierung als UML-Klassendiagramm

Alle Objekte der Klasse “AtkisMember” 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 “Polygon” für Polygone. Die folgenden 15 Zeilen zeigen eine vereinfachte Version der Deklaration der Klasse “NormalMemberType”: Zunächst wird spezifiziert, dass es sich um einen komplexen Typ handelt, indem die Schlüsselworte “complexType” (Zeile 1) und “complexContent” (Zeile 2) aus XML-Schema (Präfix “xsd”) benutzt werden. In Zeile 3 wird der GML-Typ “AbstractFeatureType” als Supertyp angegeben, wodurch alle Merkmale dieser Klasse an “NormalMemberType” vererbt werden. Der Objektidentifikator sowie die Attribute werden als Elemente deklariert (Zeilen 5 und 11) und die Geometrie als Auswahl (Zeilen 6–10).

```
01  <xsd:complexType name="NormalMemberType">
02    <xsd:complexContent>
03      <xsd:extension base="gml:AbstractFeatureType">
04        <xsd:sequence>
05          <xsd:element ref="AtkisOID" minOccurs="0"/>
06          <xsd:choice minOccurs="0">
07            <xsd:element ref="gml:location"/>
08            <xsd:element ref="gml:centerLineOf"/>
09            <xsd:element ref="gml:polygonMember"/>
10          </xsd:choice>
11          <xsd:element ref="Attribute" minOccurs="0"/>
12        </xsd:sequence>
13      </xsd:extension>
14    </xsd:complexContent>
15  </xsd:complexType>
```

Auf ähnliche Weise werden die anderen Klassen aus Bild 1 deklariert, wie “Straße”, “Weg”, “Attribute”. Dieser Struktur folgend können nun Objektexemplare angegeben werden, z.B. Straßen, Grenzen oder Verwaltungsbezirke. Die folgenden 28 Zeilen enthalten ein Beispiel dafür: die Angabe einer (sehr kurzen) Straße. Die hier aufgeführte “Badstraße” ist mit diesen Daten im EDBS-Datenbestand des DLM-Testdatenausschnitts “Türkenfeld” des Bayerischen Landesvermessungsamts enthalten (vgl. Abschnitt 6).

```
01  <AtkisMember>
02    <Strasse>
03      <gml:name> Badstrasse </gml:name>
04      <AtkisOID> 86118065 </AtkisOID>
05      <gml:centerLineOf>
06        <gml:coord> <gml:X> 4437952.980 </gml:X>
07                    <gml:Y> 5331812.550 </gml:Y> </gml:coord>
08        <gml:coord> <gml:X> 4437960.070 </gml:X>
09                    <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 Bedeutung="tatsaechliche Anzahl"> 2
16        </AnzahlDerFahrstreifen>
17        <Funktion> Strassenverkehr </Funktion>
18        <VerkehrsbedeutungInneroertlich> Anliegerverkehr
19        </VerkehrsbedeutungInneroertlich>
20        <BreiteDerFahrbahn> Keine Zuweisung </BreiteDerFahrbahn>
21        <Widmung> Gemeindestrasse </Widmung>
22        <InternationaleBedeutung> Attribut trifft nicht zu
23        </InternationaleBedeutung>
24        <VerkehrsbedeutungUeberoertlich> Attribut trifft nicht zu
25        </VerkehrsbedeutungUeberuertlich>
26      </Attribute>
27    </Strasse>
28  </AtkisMember>
```

In Zeile 1 und 2 wird angegeben, dass es sich beim folgenden Objekt um ein “AtkisMember” vom Subtyp “Strasse” handelt. Danach folgt die Angabe des Objektnamens (“Badstrasse”, Zeile 3) und des Identifikators (Zeile 4). Der Objektname ist zwar nicht ein Bestandteil der Deklaration des “NormalMemberType” weiter oben, aber er ist eine Komponente der übergeordneten GML-Klasse “AbstractFeatureCollectionType”. Die Zeilen 5–12 enthalten die Liniengeometrie der betrachteten Straße, die auch im Original tatsächlich nur drei Stützpunkte hat. Danach folgen die Straßen-Attribute mit den konkreten Werten (Zeilen 13–26), wie “Zustand: in Betrieb” oder “Widmung: Gemeindestrasse”.

Auf diese Weise könnten alle Objekte aller Objektarten eines DLM-Datenauszugs als “AtkisMember” aufgeführt werden. Beispielsweise enthält der erwähnte DLM-Testdatenausschnitt “Türkenfeld” insgesamt 5370 solcher Objekte. So kodierte Daten sind nun im Gegensatz zum ziemlich komplizierten EDBS-Datenformat sehr viel einfacher zwischen verschiedenen Benutzern auszutauschen. Außerdem gibt es XML-Werkzeuge, mit denen

man diese Daten nicht nur unverändert lesen sondern auch transformieren oder restrukturieren kann.

3 Transformation von GML-Objekten mit XSLT

Ein Werkzeug zur Transformation von XML-Dokumenten ist die XML-Sprache XSLT. Sie ist einer funktionalen Programmiersprache ähnlich und überführt XML-Dokumente – und damit auch GML-Dokumente – in XML-Dokumente anderer Struktur (*World Wide Web Consortium* 1999, *Kay* 2001). Wie die Quelldokumente, bei uns sind das mit GML kodierte DLM-Daten, sind dabei auch die Zieldokumente Bäume, und auch das jeweilige "XSLT-Programm" (Style-Sheet) hat eine Baumstruktur (vgl. Bild 2). Bei der Transformation wird nach dem Prinzip des "Pattern Matching" vorgegangen, d.h. der Quellbaum wird 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.

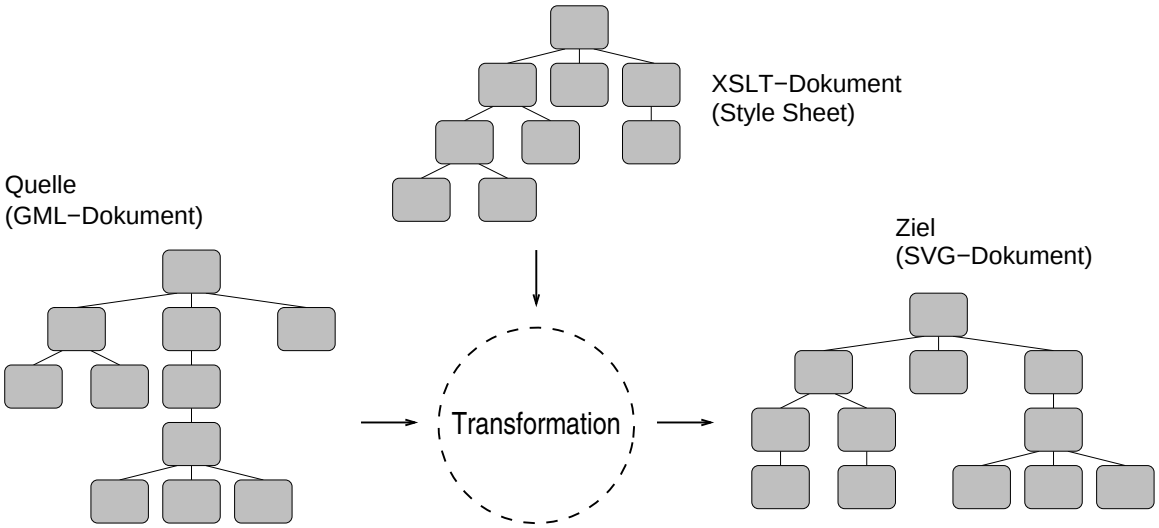

Bild 2 - Transformation mit XSLT

Wir diskutieren im Folgenden einige für unsere Anwendung wichtige XSLT-Elemente. Dabei gehen wir beispielhaft auf zwei Templates ein, die DLM-Straßenobjekte auf SVG-Zeichenbefehle mit passenden Parametern abbilden. Die Abbildung anderer linien- oder flächenförmiger Objekte geschieht analog. In den Zeilen 1–10 ist das erste XSLT-Templates aufgeführt: Innerhalb eines gegebenen GML-Quelldokuments wird nach Knoten des Typs "AtkisModell" gesucht, dort nach Konten des Typs "AtkisMember", dort nach Straßen-Konten (Zeile 2). Allgemein kann nach dem Schlüsselwort "match" ein XPath-Ausdruck stehen, und die Zeichenkette "/dlm:AtkisModell/dlm:AtkisMember/dlm:Strasse" ist ein einfaches Beispiel für solch einen Pfadausdruck. 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 3 und 4). Falls das

zutrifft, wird ein weiteres Template aufgerufen (Zeile 5), das den Namen “DrawPath” hat. Dabei wird diesem Template der Parameter “linieNebenstrasseNahverkehrVordergrund” mitgegeben (Zeilen 6 und 7). Letztlich generiert das aufgerufene Template einen SVG-Zeichenbefehl für verschiedene Arten von Linien, wobei das Aussehen der jeweiligen Linie durch den Parameter “styleclass” gesteuert wird.

```
01  <xsl:template
02       match=‘‘/dlm:AtkisModell/dlm:AtkisMember/dlm:Strasse’’>
03    <xsl:if test=‘‘contains(dlm:Attribute/dlm:Widmung,‘Gemeindestrasse’)
04                    or contains(dlm:Attribute/dlm:Widmung,‘Sonstiges’)’’>
05      <xsl:call-template name=‘‘DrawPath’’>
06        <xsl:with-param name=‘‘styleclass’’
07                    select=‘‘linieNebenstrasseNahverkehrVordergrund’’>
08      </xsl:call-template>
09    </xsl:if>
10  </xsl:template>
```

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 Kode des erwähnten Templates “DrawPath”, wobei 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), mit dem viele verschiedene Formen gezeichnet werden können, so auch alle Arten von Linien (vgl. Abschnitt 4).

```
01  <xsl:template name="DrawPath">
02    <xsl:param name="styleclass"/>
03    <svg:path>
04      <xsl:attribute name="class">
05        <xsl:value-of select="$styleclass"/>
06      </xsl:attribute>
07      <xsl:attribute name="d">
08        <xsl:for-each
09             select="gml:centerLineOf/gml:LineString/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 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 angegeben XPath-Ausdruck (Zeilen 8, 9) werden, ausgehend vom aktuellen DLM-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, das nach Gemeindestraßen sucht, auf den Knoten vom Typ "AtkisMember" aus Abschnitt 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 EDBS-Daten 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.

Insgesamt illustrieren die kurzen Beispielausschnitte wie und mit welchen Sprachmitteln GML-Dokumente mittels XSLT in andere XML-Dokumente überführt werden können: Das XSLT-Transformationsprogramm muss alle möglichen baumartigen Quelldokumente – hier: in GML kodierte DLM-Objekte – nach geeigneten Mustern durchsuchen und darauf reagierend, gewünschte Muster in die aufzubauenden Ziellbäume – hier: SVG-Dokumente – einpassen.

4 Erzeugen von kartenähnlichen Graphiken mit SVG

Wie bereits mehrfach erwähnt, ist die Zielsprache unserer Transformation der GML-Bäume die XML-Sprache SVG (Scalable Vector Graphics). SVG beschreibt zweidimensionale, verschiedenartige Grafiken in XML (vgl. z.B. *World Wide Web Consortium* 2000, *Eisenberg* 2002), wobei für uns nur die statischen Vektorgraphiken von Interesse sind. Eine der wichtigsten Anweisungen ist der bereits im letzten Abschnitt aufgetretene 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.

```
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="Pflanzensymbol"/>
```

Die obigen 7 Zeilen enthalten ein 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 Bild 3). 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 2–6): 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 3–6). In Zeile 7 wird dem Befehl – und damit dieser Signatur – ein identifizierender Name gegeben (“Pflanzensymbol”), um die Signatur in weiteren Anweisungen als Ganzes einbinden zu können.

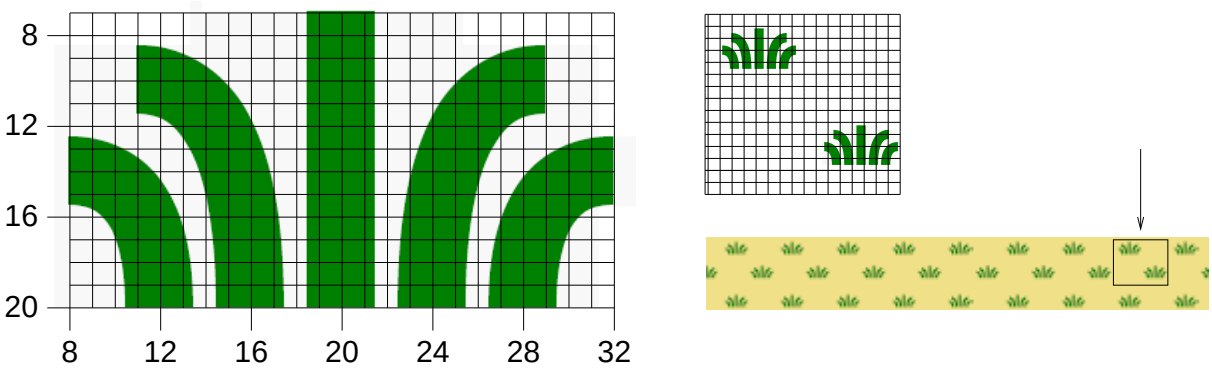

Bild 3 - Konstruktion einer Signatur und ihre Benutzung als Füllmuster

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 4 und 5). 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" width="65" y="0" x="0"/>
4    <use xlink:href="#Pflanzensymbol" transform="translate(-2,-2)"/>
5    <use xlink:href="#Pflanzensymbol" transform="translate(32,30)"/>
6  </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, 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 festlegt 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 -2794.38 949.65 -2753.24
3         919.2 -2782.51 873.62 -2831.37 925.87 -2884.03
4         961.61 -2923.33 986.69 -2899.01 1006.8 -2879.51 Z"/>
```

Damit nun alle Objekte aller Objektarten, die mit GML kodiert als Quellbaum vorliegen können, als SVG-Zeichenbefehle darstellen zu 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. Damit können, wie in Abschnitt 3 skizziert, GML-Objekte durch Mustervergleiche innerhalb von XSLT-Templates auf SVG-Zeichenbefehle abgebildet werden. Dabei wird jeweils, abhängig von der gerade betrachteten DLM-Objektart und gegebenenfalls ihrer Attribute, die passende vordefinierte Style-Definition benutzt, die ihrerseits auf vordefinierte Signaturen zurückgreift. 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.

5 Der Transformationsprozess insgesamt

Mit den in den Abschnitten 2, 3 und 4 skizzierten Elementen der Sprachen GML, XSLT und SVG haben wir ATKIS-DLM-Testdaten, die im Netz frei verfügbar sind, nun kartenähnlich visualisiert. Dabei sind wir wie folgt vorgegangen (siehe auch Bild 4): Zunächst wurden ATKIS-DLM-Testdaten im EDBS-Format aus dem Internet geladen, so wie sie von vielen Landesvermessungsämtern bereitgestellt werden. Mit dem frei verfügbaren Programm "EDBS_EXTRA" (von *Rinner* 2001) wurden diese Daten dann in ein Format umgesetzt, das sich leichter weiterverarbeiten lässt: Diese von "EDBS_EXTRA" erzeugten 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 Dateien "Attribute", "Namen" und "Linien" relativ leicht rekonstruiert werden.

Parallel zum Umformatieren der originalen EDBS-Daten wurde das in Abschnitt 2 skizzierte GML-Schema für die ATKIS-DLM-Objekte erarbeitet und schließlich gegen die GML-Spezifikation des *Open GIS Consortiums* 2001 validiert. Das Umsetzen der vorformatierten DLM-Testdaten in GML-Dokumente, die dem zuvor erarbeiteten Schema folgen, war eine größere Aufgabe. Sie wurde durch die Implementierung eines umfangreichen Java-Programmpakets 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.

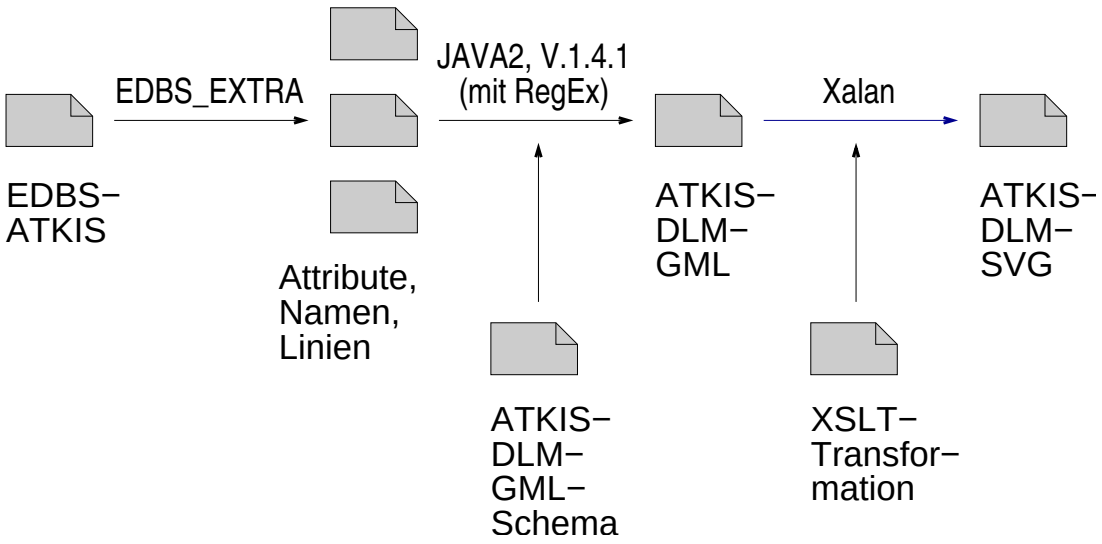

Bild 4 - der Transformationsprozess im Überblick

Diese Zieldateien sind gleichzeitig Quelldateien für die in Abschnitt 3 erläuterte Transformation der GML-Dokumente in SVG-Dokumente. Die eigentliche Transformation wurde mit dem frei verfügbaren XSLT-Prozessor "Xalan" (*Apache Software Foundation* 2003) durchgeführt; vorher musste aber natürlich die Methodik dieser Umsetzung erarbeitet und mit XSLT kodiert werden. Außerdem waren noch die in Abschnitt 4 erklärten Style-Definitionen und Signaturen zu implementieren. Zur graphischen Darstellung fertiger

SVG-Dateien auf einem Bildschirm können die üblichen Web-Browser mit einem SVG-Viewer-Plugin benutzt werden.

6 Ergebnisse und Ausblick

In der vorliegenden Fallstudie haben wir versucht, anhand realistischer Datenbestände zu zeigen, dass die XML-basierte Sprache GML gut eingesetzt werden kann, um Geo-Anwendungswelten zu modellieren, die zugehörigen Daten im entsprechenden Format zu speichern und schließlich auch kartenähnlich zu visualisieren. Dazu haben wir ein GML-Schema für ATKIS-DLM-Daten angegeben und Testdaten verschiedener Landesvermessungsämter aus dem EDBS-Format in das GML-Format überführt. Anschließend wurden diese GML-Daten mit einem XSLT-Prozessor in die XML-basierte Graphiksprache SVG transformiert. Bild 5 zeigt einen Ausschnitt der so umgesetzten DLM-Testdaten "Türkenfeld" des Bayerischen Landesvermessungsamts.

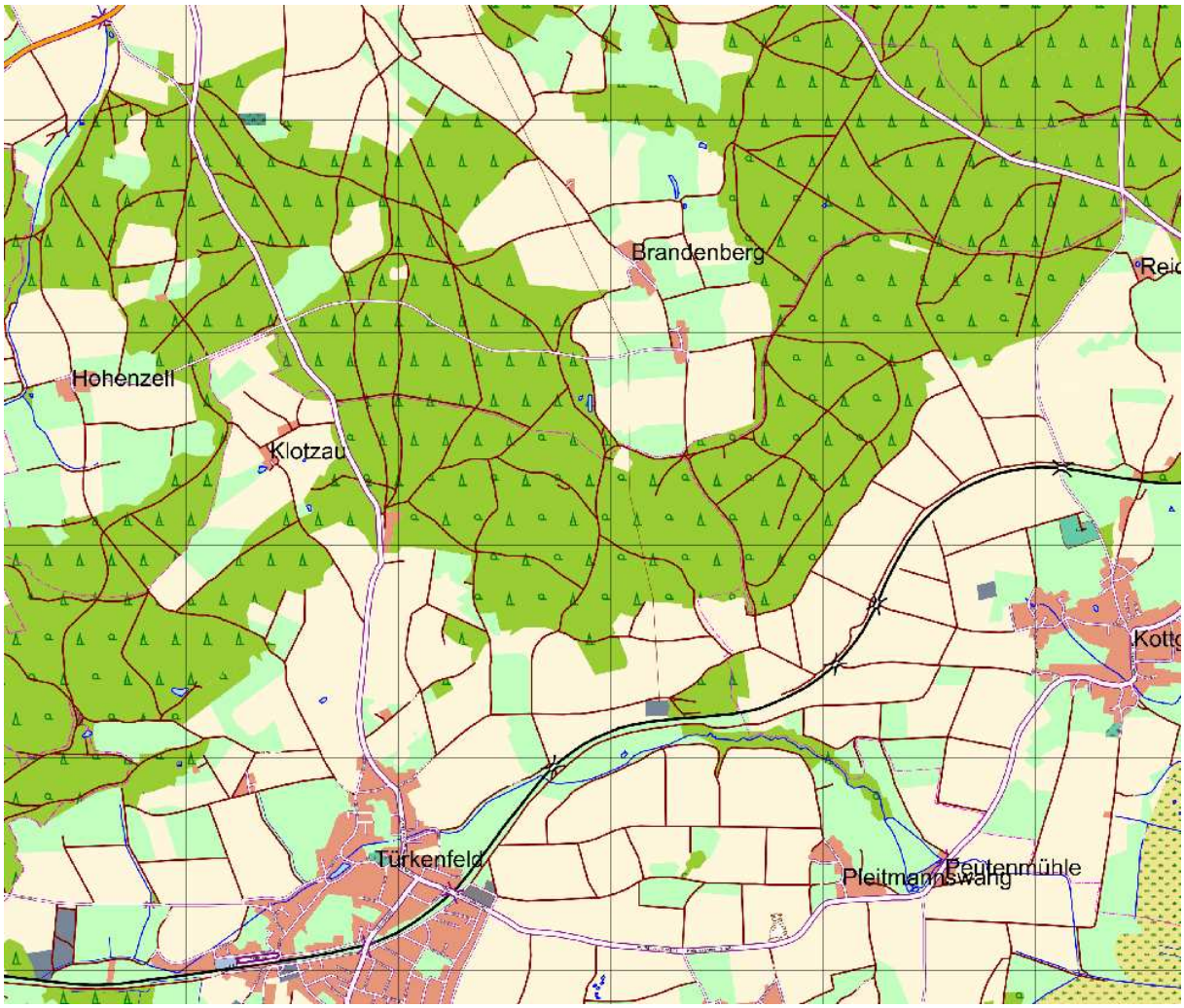

Bild 5 - generierte kartenähnliche Graphik (Ausschnitt)

Die Qualität unserer kartenähnlichen Graphiken könnte dadurch gesteigert werden, dass weitere spezifische Style-Definitionen und Signaturen implementiert würden, da wir bislang – wie erwähnt – nur ca. 50 Objektarten berücksichtigt haben. Es wäre natürlich wünschenswert, wenn es eine allgemein zugängliche Bibliothek solcher Signaturen und Style-Definitionen gäbe. Die kartographische Darstellung von GML-Daten wäre dann noch einfacher.

Literatur

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

Eisenberg, J.D: SVG Essentials. O'Reilly, 2002.

Harold, E.R.; Means, W.S.: XML in a Nutshell. O'Reilly, 2002.

Kay, M.: XSLT Programmer's Reference. 2. Auflage, Wrox Press, 2001.

Kupfer, A.: Visualisierung von GML-Daten mit XSLT und SVG. Diplomarbeit, TU Braunschweig, 2003.

Landesvermessung + Geobasisinformation Niedersachsen (Hrsg.): Bohmte, Topographische Karte 1:25000, ATKIS. 1999.

Mathiak, B.: Eine GML-Modellierung für ATKIS-DLM-Daten. Diplomarbeit, TU Braunschweig, 2003.

Neumann, K.; Eckstein, S.: Einführung in die Unified Modeling Language am Beispiel von ATKIS. In Mitteilungen des Bundesamtes für Kartographie und Geodäsie, Band 17, 2000, pp. 81–88.

Neumann, K.; Eckstein, S.: Geography Markup Language (GML) – Eine Einführung aus Informatiksicht. In Mitteilungen des Bundesamtes für Kartographie und Geodäsie, Band 24, 2003, pp. 103–111.

Open GIS Consortium (OGC): Geography Markup Language (GML) 2.0. 2001, http://www.opengis.net/gml/01-029/GML2.html.

Ray, E.T.: Einführung in XML. O'Reilly, 2001.

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

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

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