

XSLT-Werkzeuge

für GML und INTERLIS/XML

Informatik-Seminar Sommersemester 2002

Version: 9. Juli 2002

Autor: Iwan Birrer, 199b

Betreuer: Professor Stefan F. Keller

Inhaltsverzeichnis

1	INTERLIS-Beschreibungssprache und XML-Transfer	3
1.1	Einführung.....	3
1.2	Aufbau.....	3
1.3	Sprachkonstrukte.....	3
1.4	Transferformat-Ableitungsregeln	4
2	Geography Markup Language (GML)	8
2.1	Datenmodell.....	8
2.2	XML-Codierung von GML Version 2.1.1	9
2.3	GML Version 3.0	10
3	Vergleich INTERLIS/XML und GML	11
3.1	Allgemeines.....	11
3.2	Modellbasierter Ansatz.....	11
4	Extensible Stylesheet Language Transformations (XSLT)	13
4.1	Einführung.....	13
4.2	Funktionsweise	13
4.3	Syntax	14
5	Anwendungen mit XSLT und INTERLIS/XML	15
5.1	Statistik mit XSLT.....	15
5.2	Polymorphes Lesen	18
5.3	Weitere XSLT-Projekte mit INTERLIS/XML oder GML	21
	Glossar	22
	Bibliographie	23

Versionskontrolle

Änderungsgeschichte:

Datum:	Wer:	Ausgabe:	Was:
02.06.2002	ib	0.1	Entwurf/Erfassung
03.06.2002	ib	0.2	INTERLIS Beschreibung fertig
06.06.2002	ib	0.3	GML hinzugefügt
07.06.2002	ib	0.4	GML Kapitel fertig + Korrekturen
10.06.2002	ib	0.5	Korrekturen von SF Keller
11.06.2002	ib	0.6	XSLT Teil + Vergleich GML INTERLIS/XML
09.07.2002	sfk	0.7	Sprachl. Schlussredaktion

Tabelle 1-1: Änderungsgeschichte.

1 INTERLIS-Beschreibungssprache und XML-Transfer

1.1 Einführung

INTERLIS ist eine in der Schweiz spezifizierte und als Norm etablierte Datenbeschreibungssprache. Die Motivation vor über 10 Jahren eine solche Sprache zu entwerfen kommt daher, dass die Daten der amtlichen Vermessung der Schweiz einheitlich gespeichert und/oder ausgetauscht (transferiert) werden sollten.

Die Sprache soll eine möglichst präzise und vor allem eindeutige Beschreibung von Daten ermöglichen zur besseren Verständigung zwischen Fachleuten (Geomatiker, Informatiker, Manager, etc.). Zusätzlich kann die Sprache die Dokumentation von Daten vereinfachen und kann durch Wiederverwendung von Applikationen dazu beitragen, die Verarbeitung der kostspieligen zu erfassenden raumbezogenen Daten (sogenannte "Geodaten") effizienter zu verwalten.

1.2 Aufbau

INTERLIS ist grundsätzlich in zwei Teile gegliedert, die konzeptionelle Beschreibungssprache und die Spezifizierung von Ableitungsregeln (engl. encoding rules) für eine Transfersprache.

Die konzeptionelle Beschreibungssprache beschreibt ein konzeptionelles Modell einer Realität, vergleichbar mit einem UM-Modell. INTERLIS geht aber über UML hinaus und stellt Datentypen und insbesondere Konstrukte zur Verfügung, die ein eindeutiges Interpretieren des Modells zulassen. So können/müssen für jeden definierten Datentyp und jede Assoziation Konsistenzbedingungen angegeben werden. Wegen der primären Anwendung im Geoinformations-Bereich werden geometrische Basisdatentypen wie Punkte, Linien, Flächen etc. vordefiniert. Numerischen Datentypen kann eine Einheit mitgegeben werden. Die vordefinierten Datentypen sind nur nach vordefinierter Hierarchie im Sinne einer Vererbung erweiterbar.

1.3 Sprachkonstrukte

In diesem Abschnitt wird nicht detailliert auf alle Konstrukte eingegangen, es soll nur ein Überblick über die wichtigsten Konstrukte gegeben werden (detailliertere Hinweise siehe [ILIREF2.1]).

1.3.1 Klassen und Strukturen

Der Unterschied zwischen Klassen und Strukturen ist, dass Klasseninstanzen immer individuell angesprochen werden können (haben eine eindeutige Identifizierung), Strukturinstanzen nicht. Für beide könne jeweils Konsistenzbedingungen angegeben werden. Das ist z.B. UNIQUE für Werte von Attributen, die nur einmal vorkommen dürfen (mehr im Kapitel 2.11 in [ILIREF2.1]).

Ableitungen, bzw. Erweiterungen sind möglich.

1.3.2 Beziehungen

In INTERLIS wird grundsätzlich zwischen Referenzattribut und einer eigentlichen Beziehung unterschieden. Ein Referenzattribut (Referenz auf eine Klasseninstanz) kann ausschliesslich innerhalb einer Struktur vorkommen, welches so eine einseitige Beziehung beschreibt. Der Strukturinstanz ist die Klasseninstanz bekannt, umgekehrt aber nicht.

Alle anderen Beziehungen sind immer sogenannte „eigentliche Beziehungen“. In eigentlichen Beziehungen kennen sich jeweils die Beziehungspartner. Es müssen jeweils auch die Rollen, Kardinalitäten und die Stärke der Beziehung angegeben werden. Für jede eigentliche Beziehung existiert eine Beziehungsklasse (UML: Assoziative Klasse).

1.3.3 Modelle, Themen und Behälter

Jedes Datenmodell hat einen Modellnamen (Modell). Ein Modell wird z.B. für den Kanton und eines für den Bund erstellt. Modelle, bzw. deren Klassen können voneinander erben, also erweitert werden (daher wird auch von "Objektorientierung im Grossen" gesprochen).

Was in UM-Packages sind, sind in INTERLIS die Themen (Schlüsselwort Topic). Zusammengehörige Klassen werden in Themen zusammengefasst und auch so weitergegeben. Kriterien fassen in starker Beziehung stehende Klassen zusammen und werden analog zu Zuständigkeiten der enthaltenen Daten abgegrenzt: z.B. Tiefbauamt => Thema Strassen; Gemeindewerke => Thema Wasserleitungen; Topics können im Gegensatz zu Packages nicht geschachtelt werden. Bei der Instanziierung, bzw. beim Transfer, werden die Themen zu Behältern (engl. baskets) und erhalten selber eine eindeutige Identifizierung.

1.4 Transferformat-Ableitungsregeln

1.4.1 Allgemeines

INTERLIS schreibt nicht von vorneherein ein fixes Transferformat vor, es gibt jedoch klare Ableitungsregeln für anwendungsspezifische Schemas. Damit ist das Transferformat für jedes Anwendungsschema bestimmt. Es wird zwar in der Version 2.1 von INTERLIS nur der Transfer mit XML 1.0 speziell beschrieben, es gibt aber einige Grundregeln, die für alle möglichen Arten von Transferformaten gültig sein müssen. Als Alternativen werden genannt CORBA, DCOM (.NET) und Web Services [ILIREF2.1] (Kapitel 3.1).

Die allgemeinen Eigenschaften der Transferformat-Ableitungsregeln sind hier kurz zusammengefasst:

- Polymorphes Lesen muss unterstützt werden ([ILIREF2.1] Kapitel 3.2.2).
- Inkrementelle Nachlieferung von Daten muss unterstützt werden. Das heisst eine Datensinke muss mit Daten erweitert werden können, ohne dass jedes Mal die gesamten Daten geschickt werden müssen.
- Der Aufbau eines Transfers ist stets ein Vorspann gefolgt von einem Datenbereich.

Der Vorspann und der Datenbereich sind noch weiter spezifiziert, z.B. die Reihenfolge der Klasseninstanzen ist vorgeschrieben ([ILIREF2.1] Kapitel 3.2.5 bis 3.2.8).

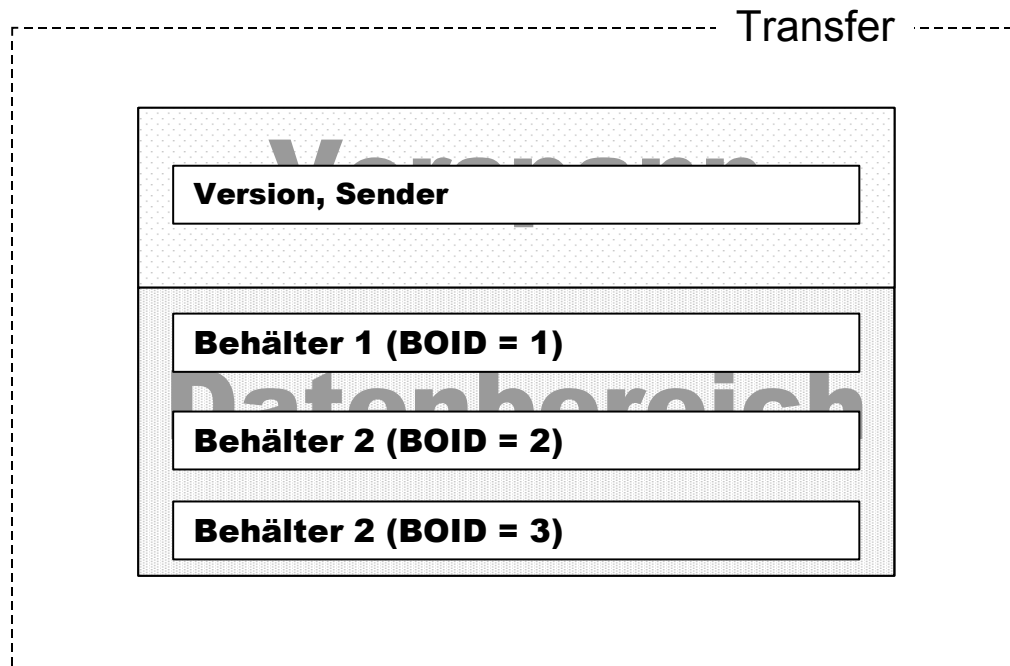


Abbildung 1-1: Ein Transfer ist in INTERLIS grundsätzlich in die beiden Bereiche Vorspann und Datenbereich aufgeteilt.

1.4.2 XML-Transferformat

1.4.2.1 Aufbau

In XML sieht der grundlegende Aufbau folgendermassen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<TRANSFER xmlns:xsi="..." xsi:noNamespaceSchemaLocation="...">
  <HEADERSECTION>
    <!--
      Angaben über die Version und den Sender der Daten.
      Thema/Klassen/Attribut - Auflösungen für Ploymorphes lesen.
    -->
  </HEADERSECTION>
  <DATASECTION>
    <!--
      Hier kommen die eigentlichen Daten.
      Als Sequenz von Behältern.
    -->
    <%Modellname.Themenname% BOID = "ID1">
      <!--
        Hier kommen die Objektinstanzen: Codiert nach dem Schema
      <%Modellname.Themenname.Klassenname% TID = "ID">
        Für die inkrementelle Nachlieferung muss anstatt eines TID
        Attributs ein TOID Attribut angegeben werden, welcher immer
        eindeutig sein muss.
      -->
    </%Modellname.Themenname%>
    <%Modellname.Themenname% BOID = "ID2">
```

```

        . . . . .
    </DATASECTION>
</TRANSFER>
    
```

1.4.2.2 Behälter und Objekte

Die Codierung erfolgt immer auf der Basis von Behältern. Im Datenbereich ist die oberste Hierarchiestufe also ein Behälter. Jeder Behälter muss eine eindeutige Identifikation haben, codiert als BOID (Basket Object Identifier) Attribut. Das XML-Tag für einen Behälter ist in den Beispielen aus didaktischen Gründen nach folgendem Muster codiert: Modellname.Themename. Für eine Objektinstanz sieht das ganz ähnlich aus: Modellname.Themename.Klassenname. Für echte Anwendungen sind ein Standardregeln für schweizweite eindeutige OIDs empfohlen (siehe [ILIREF2.1] Anhang E Aufbau von Objektidentifikatoren (OID)).

1.4.2.3 Polymorphes Lesen

Wie von den allgemeinen Regeln vorgeschrieben baut auch das XML Format auf einem Vorspann und einem Datenbereich auf. Der Vorspann dient unter anderem zur Unterstützung des polymorphen Lesens. Um das zu ermöglichen, werden für jedes Modell, das in der konzeptionellen Beschreibungssprache vorkommt, Angaben über die Auflösung von Themen, Klassen und Attributen in ihre Oberthemen, Oberklassen und Oberattribute gemacht. Kommt z.B. im Modell B das Topic B vor, welches vom Topic A abgeleitet ist, so muss bei einem Transferfile für das Modell B stehen, dass das Topic B in das Topic A aufgelöst werden soll.

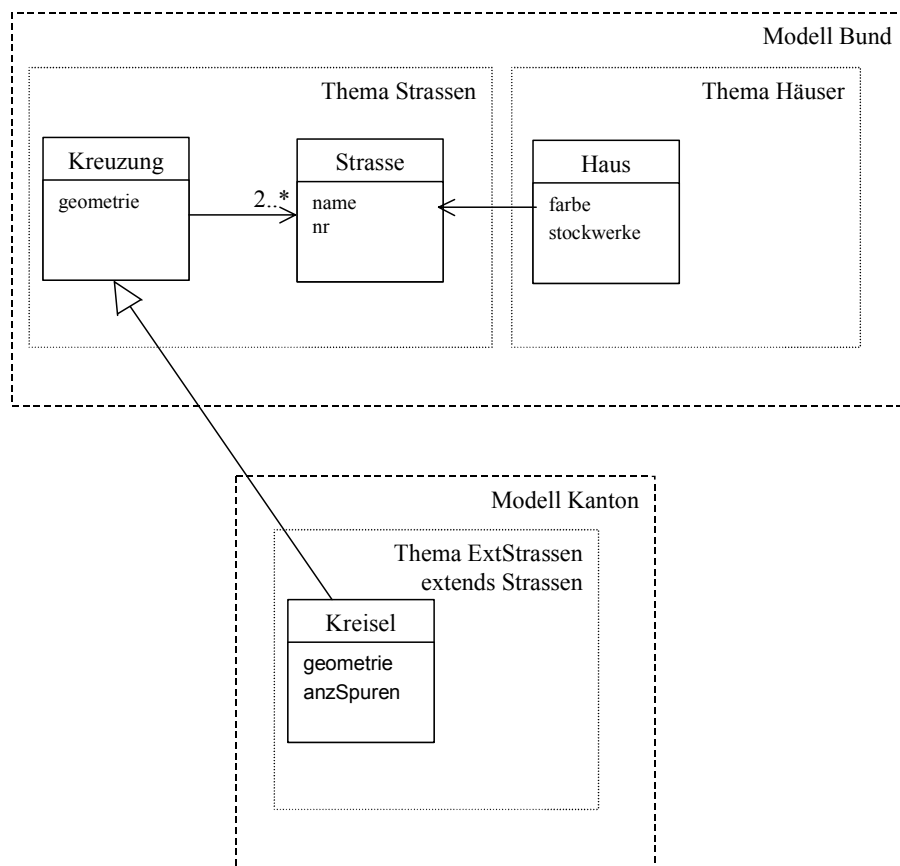


Abbildung 1-2: Polymorphes Lesen von Daten aus erweiterten Datenmodellen (Modell Kanton).

Beim folgenden Beispiel müsste mal also für einen Vorspann für das Modell Kanton folgende Auflösungen angeben:

- ExtStrassen -> Strassen
- Kreisel -> Kreuzung
- Kreisel.anzSpuren -> löschen

Nun kann ein Tool, das nur das Modell Bund kennt trotzdem Daten vom Modell Kanton lesen. Kreisel werden dann zwar nicht als solche dargestellt, wohl aber als eine Kreuzung.

1.4.2.4 Beziehungen

Eigentliche Referenzen werden in INTERLIS 2.1 immer mit einem Beziehungsobjekt gespeichert. Das Beziehungsobjekt enthält für jede Rolle der Beziehung ein XML Tag mit dem Rollennamen.

Zukünftige Versionen von INTERLIS werden das codieren:

```
<Bund.Haeuser.StrassenAssoziation>  
  <RolleHaus REF="10" />  
  <RolleStrasse BOID="1" EXTREF="5" />  
</Bund.Haeuser.StrassenAssoziation>
```

Das hat den Vorteil, dass das Beziehungsobjekt als solches identifiziert werden kann, weil ein REF- bzw. ein EXTREF-Attribut vorkommt.

Die BOID (Behälter-Objekt-Identifikation) muss nur angegeben werden, wenn sich das Objekt in einem anderen Behälter (welcher nicht zwingend von einem anderen Thema sein muss) befindet. Hier ist dies der Fall, da sich die Rolle `RolleStrasse` in einem anderen Behälter (bzw. Thema) befindet.

2 Geography Markup Language (GML)

GML ist sozusagen das amerikanische Pendant zu INTERLIS. GML ist mehr auf eigentliche Raumdaten ausgerichtet und nicht unbedingt von vornherein als allgemeine Datenbeschreibungssprache konzipiert.

GML ist nur *eine* Implementationspezifikation für XML ([GML_SPEC2.1.1]) von einer allgemeinen sogenannten "abstrakten Spezifikation", welche nicht nur die Beschreibung, wie Geodaten ausgetauscht werden sollen beinhaltet, sondern z.B. auch welche Operationen (Funktionen) auf die vorhandenen Daten angewendet werden können. Zudem ist die abstrakte Spezifikation völlig losgelöst von einer Implementierung (Programmiersprache, Betriebssystem, Datenbankart etc.) und ist so langlebiger als konkrete Implementierungen. Von der abstrakten Spezifikation gibt es auch eine Implementation für SQL und .

GML 2.1 ist nur eine Untermenge der Implementation der abstrakten Spezifikation.

Da GML vor allem in den USA Verbreitung findet, ist sie auch auf die dortigen Gegebenheiten ausgerichtet, bzw. von deren Industrievertretern bestimmt (gilt für zukünftige Versionen 3 bzw. 4 nur noch bedingt).

GML und die abstrakte Spezifikation wurden von einem Konsortium „Open GIS Consortium“ genannt geschrieben. Das OGC ist eine Non-Profit Organisation und in ihrem Aufbau vergleichbar mit der W3C (World Wide Web Consortium). Es gibt sogenannte Working Groups die eine Spezifikation ausarbeiten und ein technisches Komitee, welches die Spezifikation als Standard verabschiedet.

2.1 Datenmodell

Die zurzeit aktuelle Version von GML ist 2.1.1. Anfragen haben aber ergeben, dass bereits GML 3.0, bzw. sogar schon GML 4.0 in Diskussion sein soll. Zukünftige Versionen werden die Implementation um Funktionen erweitern, welche z.T. bereits in der abstrakten Spezifikation spezifiziert sind - bei Bedarf wird einfach die abstrakte Spezifikation angepasst (etwas, das in letzter Zeit häufiger vorkam, als erwartet wurde, da z.B. Kompromisse durch die Zusammenarbeit mit der internationalen Normierungsverband ISO eingegangen werden mussten).

Zur Zeit hat GML vor allem eine Einschränkung. Es werden nur Geometrie-Datentypen mit linear verbundenen Linien unterstützt, also keine Kreissegmente, Splines etc. Genau dies soll in GML 3 vorhanden sein (siehe Kapitel 2.3).

Die Spezifikation von GML und allgemein die ganze OGC braucht durchgehend UML als Beschreibungssprache für Daten-Modelle - jedoch nur als Arbeits- und Dokumentationsinstrument und nicht als Ausgangslage um daraus automatisch GML abzuleiten. Das wird mit GML 4 angestrebt, wobei nicht sicher ist, ob die Mehrheit innerhalb des OGC Verständnis dafür aufbringt. Um ein Datenmodell konform zum GML-Standard zu erstellen muss man ein paar grundlegende Regeln einhalten.

- Jede Klasse die man erstellt, muss (direkt oder indirekt) entweder von der GML-Klasse `AbstractFeatureType` oder `AbstractGeometryType` abgeleitet sein.

- Beziehungen zwischen den Klassen können jeweils eine assoziative Klasse angehängt haben, welche die Beziehung genauer beschreibt.
- Restriktionen für Attribute, Klassen und Beziehungen werden in einem XML Schema ausgedrückt.

Die genauen Regeln findet man in der GML Implementation Specification 2.1.1 ([GML_SPEC2.1.1]) im Kapitel 4 und 5.

Das gleiche Datenmodell wie oben würde in GML etwa so aussehen:

Abbildung 2-3: Datenmodell in GML.

2.2 XML-Codierung von GML Version 2.1.1

Die XML Codierung baut auf den folgenden Standards von W3C auf:

[XML1.0]	Grundlegende XML Spezifikation Version 1.0.
[XML_NAMES]	GML macht Gebrauch von XML Namespaces um Namensräume für das GML-Datenmodell (gml:) und für neue Datenmodelle zu definieren.
[XML_SCHEMA]	GML braucht das XML Schema um den Aufbau einer XML-Datei eines Datenmodells zu beschreiben. Alle Daten die in GML codiert werden müssen immer ein gültiges XML Schema angehängt haben.
[XML_LINK1.0]	Um Beziehungen zwischen Klassen (Features) zu beschreiben, macht GML Gebrauch von dem XLink-Standard. Somit können alle Arten von Beziehungen beschrieben werden.

[XML_POINTER1.0] Die XML Pointer-Spezifikation wird von der XLink-Spezifikation verwendet um auf einzelne Elemente in einer verlinkten XML-Datei zugreifen zu können.

2.3 GML Version 3.0

Zum jetzigen Zeitpunkt ist steht noch keine Dokumentation bzw. Spezifikation von GML 3.0 zur Verfügung. Ein provisorisches XML Schema ist allerdings schon verfügbar und man kann daraus einige Schlüsse auf die Version 3.0 schliessen.

Neu gibt es nicht mehr nur 2 grundlegende Schemas sondern die Schemas sind in verschiedene Module aufgeteilt.

Es werden Units eingeführt, um die Mass-Einheiten von numerischen Typen zu erfassen. Ebenfalls werden Geometrie-Datentypen eingeführt, welche auch nicht lineare Verbindungen zwischen zwei Kurven zulassen (z.B. ArcType). An der allgemeinen Codierung wird sich aber kaum etwas ändern, so sollte es kein Problem sein Daten im GML 2.1.1 Format auch in Applikationen mit GML 3.0 Unterstützung zu lesen.

3 Vergleich INTERLIS/XML und GML

3.1 Allgemeines

GML und INTERLIS/XML sind beides Codierungssprachen, die auf die Beschreibung von Raumdaten spezialisiert sind.

3.2 Modellbasierter Ansatz

Man kann INTERLIS/XML und GML nicht nur aufgrund des Transferformates vergleichen, sondern muss vor allem miteinbeziehen, wie die Formate entstanden sind.

Der grösste Unterschied liegt darin, dass INTERLIS ein eindeutiges konzeptionelles Modell zugrunde liegt (modellbasierter Ansatz). Eindeutig heisst, dass genau festgelegt ist, was z.B. eine Aggregation ist und was mit den enthaltenen Objekten passiert, falls die enthaltende Klasse kopiert wird, etc. (siehe [ILIREF2.1] Kapitel 1). Es wurde einmal ein Modell erstellt, auf das sich alle Implementationen (sei das in XML, SQL etc.) abstützen. INTERLIS hat also von vornherein einen Top-down-Ansatz gewählt und auch konsequent durchgezogen.

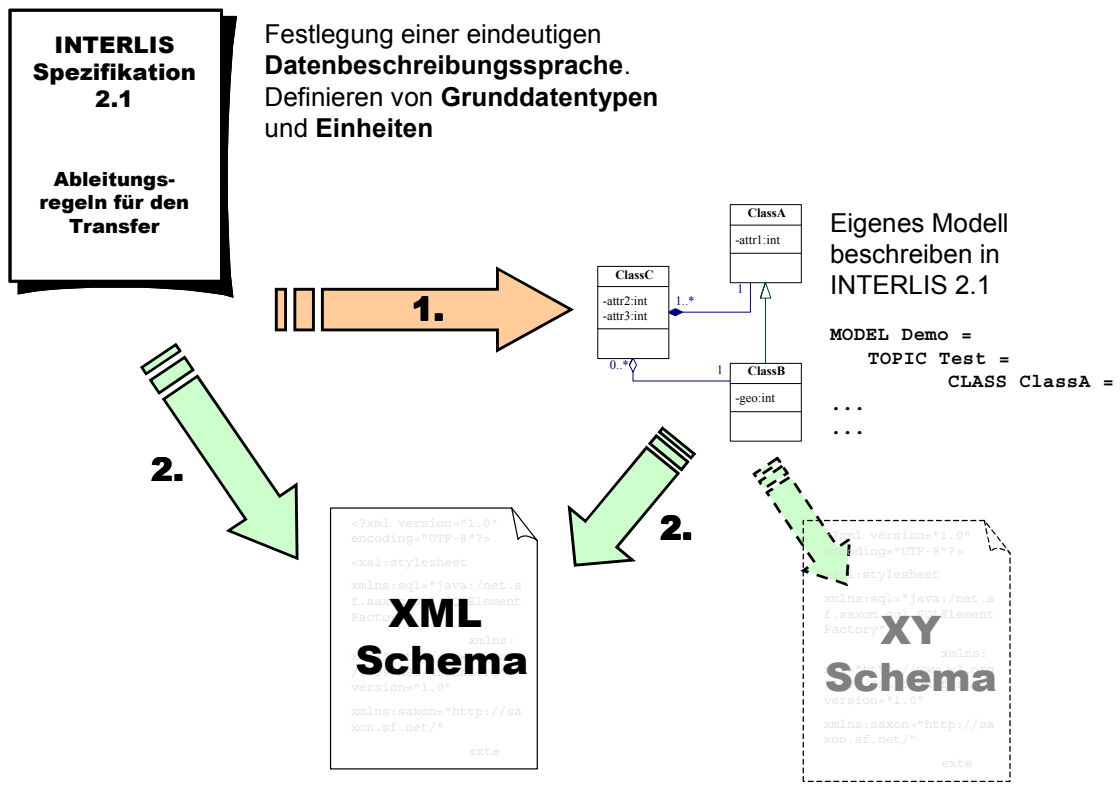


Abbildung 3-4: INTERLIS 2.1: Weg von der Spezifikation bis zum XML-Schema, welches die Codierung schlussendlich definiert. Die grünen Pfeile (2. Schritt) zeigen an, dass dieser Schritt aus Regeln ableitbar ist, also automatisch durchgeführt werden kann.

GML liegt zwar mit der Abstract Spezifikation auch ein Modell zugrunde, dieses ist aber erstens nicht eindeutig definiert (keine genaue Regeln) und zweitens nicht konsequent durchgezogen. So ist wird in der GML Implementation Specification zum Beispiel das in der Abstract Specification modellierte Geometriemodell eingeschränkt. Um ein eigenes Modell zu definieren wird in GML XML-Schema verwendet, wobei man von vorgegebenen Schemas ableiten muss. Dieses Schemas ist aber nur für den Transfer mit XML von Nutzen. Man ist an die Grenzen des XML Schema ([XML_SCHEMA]) Standards gebunden. Es gibt also kein genau definiertes eigenes Datenmodell.

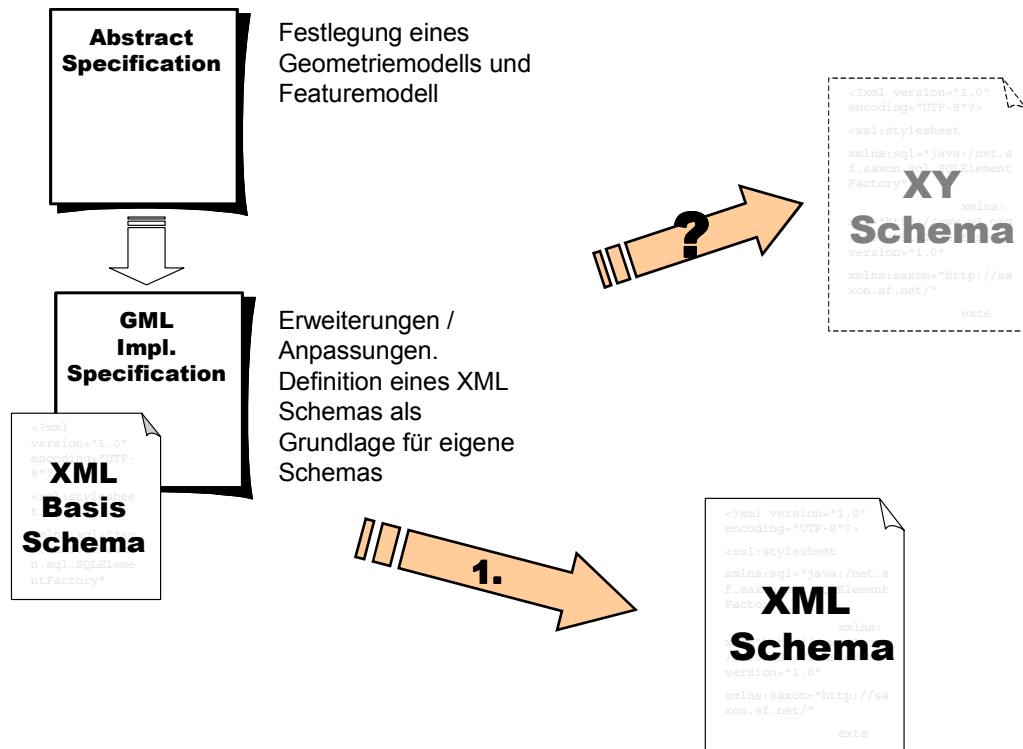


Abbildung 3-5: GML 2.1.1: Weg von der Spezifikation bis zum XML-Schema. Es taucht nirgends ein genau definiertes Modell auf.

Beziehungen werden in GML mit XLINK realisiert, einer Spezifikation der W3C. Daraus resultiert allenfalls ein Vorteil für GML, sodass generisch für XLINK programmierte Tools für GML eingesetzt werden können. Allgemein stützt sich GML mehr auf die zusätzlichen Spezifikationen vom W3C ab als INTERLIS/XML.

Grunddatentypen sind in INTERLIS (bis auf wenige Ausnahmen) nicht als Klassen, bzw. Strukturen definiert, sondern haben eine eigene Beschreibungssprache. Für jemanden, der XML Schema bereits kennt, ist es deshalb nicht ganz einfach, diese Datentypen in XML vorzustellen und abzubilden. Es gibt aber eindeutige Regeln wie man einen Grunddatentyp in XML darstellt.

GML hat es da einfacher, da das ganze Schema (also auch Grunddatentypen) auf Klassen aufbaut.

4 Extensible Stylesheet Language Transformations (XSLT)

4.1 Einführung

XSLT ist eine auf XML 1.0 basierte Sprache, um Regeln auszudrücken wie ein XML-Dokument in irgendein anderes Textdokument (meist XML) überführt werden kann.

Die Spezifikation für XSLT ([XML_XSLT1.0]) wurde bereits Ende 1999 als normativer Standard vom W3C verabschiedet. XSLT 2.0 ist zurzeit noch ein „Working Draft“ ([XML_XSLT2.0]). Hinzukommen wird in Version 2.0 vor allem die Unterstützung für XML Schema.

4.2 Funktionsweise

Um die Regeln die in einem XSLT Dokument beschreiben sind, auf ein XML-Dokument anzuwenden, braucht es einen XSLT-Prozessor. Bekannte und frei erhältliche sind unter anderem:

- Saxon (<http://saxon.sourceforge.net/>); unterstützt versuchsweise XSLT 2.0
- Xalan (<http://xml.apache.org/xalan-j/index.html>)
- MSXML (<http://www.microsoft.com>)

Als Input verlangt der XSLT-Prozessor eine XSLT-Datei und eine XML-Datei. Der Output ist eine Textdatei, die wiederum eine XML-Datei sein kann.

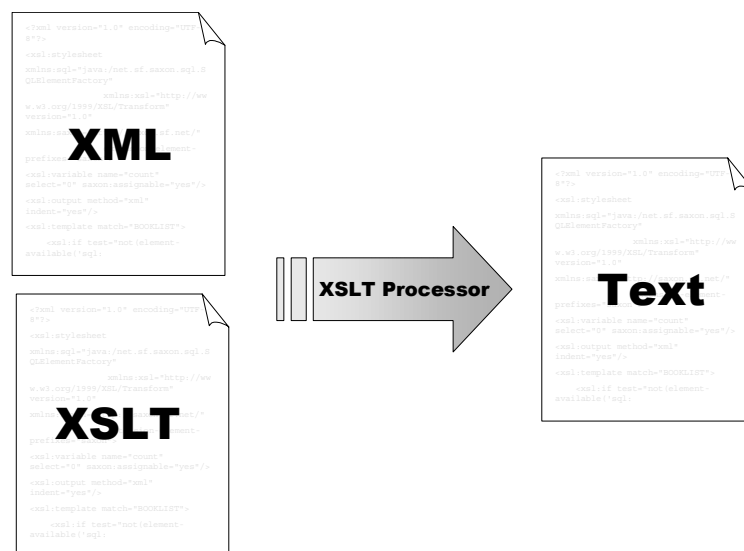


Abbildung 4-6: Ablauf einer XSLT Transformation.

4.3 Syntax

Alle Syntaxregeln hier zu erklären, würde den Umfang diese Berichts sprengen. Darum hier nur einige Grundregeln:

- Um einen spezifischen Ast in einem XML - Dokument anzusprechen wird XPATH gebraucht (siehe: [XML_XPATH1.0])
- Es gibt Befehle um ganze Äste auszugeben:

```
<xsl:copy-of select="[XPath expression]">
```

oder nur Text Nodes auszugeben: :

```
<xsl:copy-of select="[XPath expression]">
```

- Man kann mit Templates einen Ast aus der Source auswählen (match) und ihn in einen anderen überführen (alles was innerhalb des Template Tags steht)

```
<xsl:template match="[XPath expression]">
```

```
  <Neuer-Ast>
```

```
    <xsl:copy-of select="[XPath expression]"> <!-- Elemente vom  
                                     Source Tree hineinnehmen -->
```

```
  </Neuer-Ast>
```

```
</xsl:template>
```

Informationen zu XSLT (Einführungen, Tutorials, Foren, etc.) ist in der Bibliographie zu finden.

5 Anwendungen mit XSLT und INTERLIS/XML

5.1 Statistik mit XSLT

Im Sprachumfang von XSLT gibt es die Funktion `count([XPath expression])`, mit welcher man Elemente in einem XML-Dokument zählen kann. Das kann man z.B. verwenden, um eine Statistik einer INTERLIS/XML-Datei zu generieren.

Folgendes Beispiel zählt alle Behälter und Objekte einer INTERLIS/XML-Datei. Die Grafik zeigt den Aufbau des Skripts:

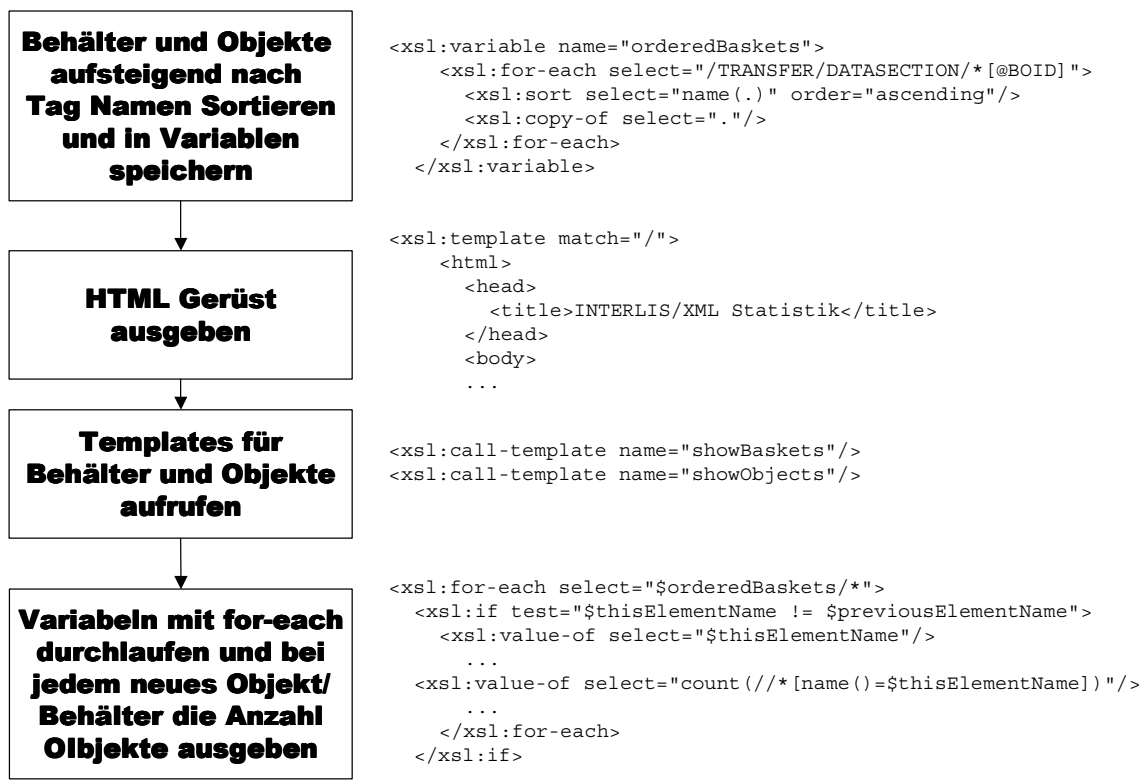


Abbildung 5-7: Ablauf des XSLT-Statistik-Skripts.

Das XSLT-Skript:

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:variable name="totalNrOfBaskets"
  select="count (/TRANSFER/DATASECTION/*[@BOID])"/>
  <xsl:variable name="orderedBaskets">
    <xsl:for-each select="/TRANSFER/DATASECTION/*[@BOID]">
      <xsl:sort select="name()" order="ascending"/>
      <xsl:copy-of select="."/>
    
```

```

    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="orderedObjects">
    <xsl:for-each select="/TRANSFER/DATASECTION/*[@BOID]/*[@TID]">
      <xsl:sort select="name(.)" order="ascending"/>
      <xsl:copy-of select="."/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="totalNrOfObjects"
select="count (/TRANSFER/DATASECTION/*[@BOID]/*[@TID])"/>
  <xsl:variable name="ob" select="$orderedBaskets"/>
  <xsl:variable name="oo" select="$orderedObjects"/>
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <title>INTERLIS/XML Statistik</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
        <link href="main.css" rel="stylesheet" type="text/css"/>
      </head>
      <body>
        <table id="stufe1" width="100%" border="0" cellspacing="0"
cellpadding="5">
          <tr>
            <th>INTERLIS/XML Statistik</th>
          </tr>
          <tr>
            <td>
              <p>Modulname:
                <xsl:value-of select="substring-
before (name (/TRANSFER/DATASECTION/*[1]), '.')"/>
              </p>
              <p>Kommentar:
                <xsl:value-of select="/TRANSFER/HEADERSECTION/COMMENT"/>
              </p>
              <xsl:call-template name="showBaskets"/>
              <xsl:call-template name="showObjects"/>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="showPolyline"/>
  <xsl:template name="showBaskets">
    <table id="stufe2" width="100%" border="0" cellspacing="0" cellpadding="4">
      <tr>
        <th>
          <p>Behaelter (Total: <xsl:value-of select="$totalNrOfBaskets"/>)</p>
        </th>
      </tr>
      <tr>
        <td class="formElementLegend">
          <table id="stufe3" width="100%" border="0" cellspacing="0"
cellpadding="3">
            <tr>
              <th width="250">Name</th>
              <th>Anzahl</th>
            </tr>
            <xsl:for-each select="$ob/*">
              <xsl:variable name="pos" select="position()"/>
              <xsl:variable name="previousElementName" select="name(../*[$pos -
1])"/>

```

```

        <xsl:variable name="thisElementName"
select="name(self::node())"/>
        <xsl:variable name="thisElement" select="self::node()"/>
        <xsl:if test="$thisElementName != $previousElementName">
            <tr>
                <td>
                    <xsl:value-of select="substring-after( $thisElementName,
'.')"/>
                </td>
                <td align="right">
                    <xsl:value-of
select="count (//*[name()=$thisElementName])"/>
                </td>
            </tr>
        </xsl:if>
    </xsl:for-each>
</table>
</td>
</tr>
</table>
</xsl:template>
<xsl:template name="showObjects">
    <table id="stufe2" width="100%" border="0" cellpadding="4">
        <tr>
            <th>
                <p>Objekte (Total: <xsl:value-of select="$totalNrOfObjects"/>)</p>
            </th>
        </tr>
        <tr>
            <td class="formElementLegend">
                <table id="stufe3" width="100%" border="0" cellpadding="3">
                    <tr>
                        <th width="250">Name</th>
                        <th>Anzahl</th>
                    </tr>
                    <xsl:for-each select="$oo/*">
                        <xsl:variable name="pos" select="position()"/>
                        <xsl:variable name="previousElementName" select="name(../*[$pos -
1])"/>
                        <xsl:variable name="thisElementName"
select="name(self::node())"/>
                        <xsl:variable name="thisElement" select="self::node()"/>
                        <xsl:if test="$thisElementName != $previousElementName">
                            <tr>
                                <td>
                                    <xsl:value-of select="substring-after( substring-after(
$thisElementName, '.'), '.')/>
                                </td>
                                <td align="right">
                                    <xsl:value-of
select="count (//*[name()=$thisElementName])"/>
                                </td>
                            </tr>
                        </xsl:if>
                    </xsl:for-each>
                </table>
            </td>
        </tr>
    </table>
</xsl:template>
</xsl:stylesheet>

```

Nachfolgend ein Bildschirm-Ausdruck mit den Daten des Roads-Beispiels aus INTERLIS 2:

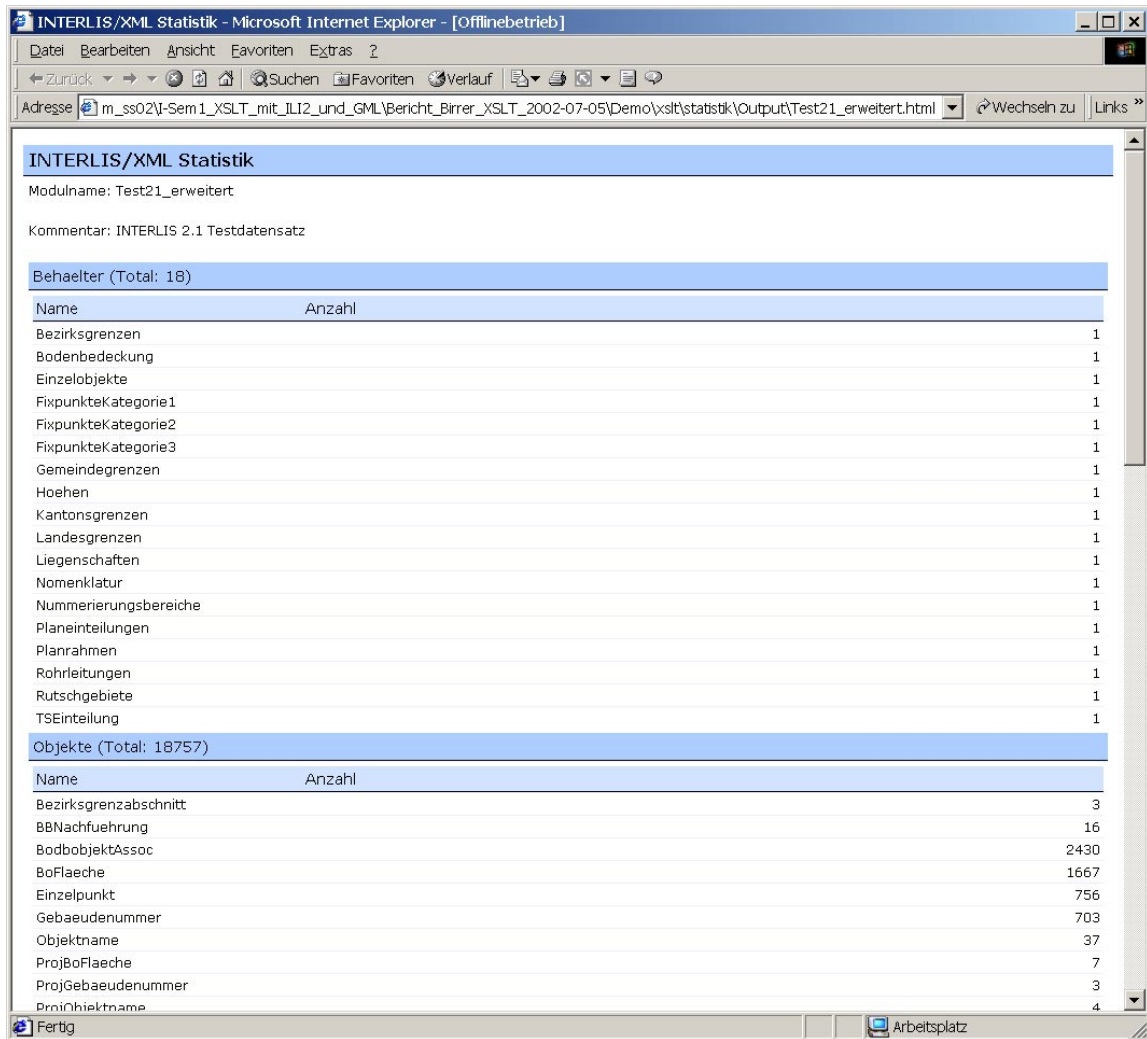


Abbildung 5-8: Bildschirm-Ausdruck des Statistik-Skripts mit den Daten des Roads-Beispiels.

5.2 Polymorphes Lesen

Am meisten wird XSLT gebraucht, um ein XML-Dokument in einem Format (passend zu einem bestimmten XML Schema) in ein XML-Dokument eines anderen Formats zu konvertieren. Genau das muss gemacht werden, wenn man ein INTERLIS/XML-Dokument eines Untermodelles in ein INTERLIS/XML eines Obermodelles umwandeln möchte.

INTERLIS stellt schreibt die Informationen für die Umwandlung in ein Obermodell direkt in den Vorspann des XML-Dokumentes.

Das folgende XSLT-Skript nimmt diese Umwandlung vor:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:param name="targetModel" select="'RoadsExdm2ben_10'"/>
```

```

<xsl:variable name="entries"
select="/TRANSFER/HEADERSECTION/ALIAS/ENTRIES[@FOR=string($targetModel)]"/>
<xsl:template match="/TRANSFER">
  <!--Polymorphic Reader for INTERLIS 2.1-->
  <xsl:if
test="count(/TRANSFER/HEADERSECTION/ALIAS/ENTRIES[@FOR=string($targetModel)])!=0">
    <TRANSFER>
      <HEADERSECTION VERSION="2.1" SENDER="{HEADERSECTION/@SENDER}">
        <ALIAS>
          <!-- <xsl:copy-of
select="child::HEADERSECTION/child::ALIAS/child::ENTRIES[@FOR=string($targetModel)]
/preceding-sibling::*"/>
          <xsl:copy-of
select="child::HEADERSECTION/child::ALIAS/child::ENTRIES[@FOR=string($targetModel)]
"/> -->
          <xsl:comment>
Here the Alias Table Entries come in!!
Build it with the INTERLIS Compiler and paste it here
          </xsl:comment>
          </ALIAS>
        </HEADERSECTION>
      <DATASECTION>
        <xsl:apply-templates select="/TRANSFER/DATASECTION"/>
      </DATASECTION>
    </TRANSFER>
  </xsl:if>
<xsl:if
test="count(/TRANSFER/HEADERSECTION/ALIAS/ENTRIES[@FOR=string($targetModel)])=0">
  <xsl:message terminate="yes">The specified model: '<xsl:value-of
select="$targetModel"/>' could not be found.</xsl:message>
  </xsl:if>
  <xsl:message>Transformation done.</xsl:message>
</xsl:template>
<!-- DATASECTION TEMPLATES -->
<!-- Template for Objects with a TID -->
<xsl:template match="*[@BOID]/*[@TID]">
  <xsl:variable name="longName" select="name(.)"/>
  <!--Original Name ist:<xsl:value-of select="$longName"/>-->
  <xsl:variable name="alias">
    <!-- Kein Eintrag für diesen Tag -->
    <xsl:choose>
      <!-- Entry in TAGENTRY found? -->
      <xsl:when test="count($entries/TAGENTRY[@FROM=string($longName)])!=0">
        <xsl:value-of
select="$entries/TAGENTRY[@FROM=string($longName)]/@TO"/>
      </xsl:when>
      <!-- Entry in DELENTY found? -->
      <xsl:when test="count($entries/DELENTY[@FROM=string($longName)])!=0">
        <!-- Do Nothing Set alias to null -->
      </xsl:when>
      <!-- ELSE -->
      <xsl:otherwise>
        <xsl:value-of select="$longName"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <!--Alias gesetzt auf:<xsl:value-of select="$alias"/>-->
  <!-- if alias entry has NULL entry, discard current object -->
  <xsl:if test="$alias!=string('')">
    <xsl:element name="{ $alias }">
      <xsl:attribute name="TID"><xsl:value-of select="@TID"/></xsl:attribute>
      <xsl:if test="@BOID!=string('')">

```

```

        <xsl:attribute name="BOID"><xsl:value-of
select="@BOID"/></xsl:attribute>
    </xsl:if>
    <xsl:if test="@OPERATION!=string('')">
        <xsl:attribute name="OPERATION"><xsl:value-of
select="@OPERATION"/></xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
</xsl:element>
</xsl:if>
</xsl:template>
<xsl:template match="*[@BOID]">
    <xsl:variable name="longName" select="name(.)"/>
    <!--Original Name ist:<xsl:value-of select="$longName"/>-->
    <xsl:variable name="alias">
        <!-- Kein Eintrag für diesen Tag -->
    <xsl:choose>
        <!-- Entry in TAGENTRY found? -->
        <xsl:when test="count($entries/TAGENTRY[@FROM=string($longName)])!=0">
            <xsl:value-of
select="$entries/TAGENTRY[@FROM=string($longName)]/@TO"/>
        </xsl:when>
        <!-- Entry in DELENTY found? -->
        <xsl:when test="count($entries/DELENTY[@FROM=string($longName)])!=0">
            <!-- Do Nothing Set alias to null -->
        </xsl:when>
        <!-- ELSE -->
        <xsl:otherwise>
            <xsl:value-of select="$longName"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:variable>
    <!--Alias gesetzt auf:<xsl:value-of select="$alias"/>-->
    <!-- if alias entry has NULL entry, discard current object -->
    <xsl:if test="$alias!=string('')">
        <xsl:element name="{ $alias }">
            <xsl:attribute name="BOID"><xsl:value-of select="@BOID"/></xsl:attribute>
            <xsl:if test="@KIND!=string('')">
                <xsl:attribute name="KIND"><xsl:value-of
select="@KIND"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="@STARTSTATE!=string('')">
                <xsl:attribute name="STARTSTATE"><xsl:value-of
select="@STARTSTATE"/></xsl:attribute>
            </xsl:if>
            <xsl:if test="@ENDSTATE!=string('')">
                <xsl:attribute name="ENDSTATE"><xsl:value-of
select="@ENDSTATE"/></xsl:attribute>
            </xsl:if>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:if>
</xsl:template>
<xsl:template match="*[@BOID]/*[@TID]/*">
    <xsl:variable name="name" select="name(.)"/>
    <xsl:variable name="longName">
        <xsl:value-of select="name(..)"/>.<xsl:value-of select="$name"/>
    </xsl:variable>
    <xsl:variable name="value" select="text()"/>
    <!--Original Name ist:<xsl:value-of select="$longName"/>-->
    <xsl:variable name="alias">
    <xsl:choose>

```

```

<!-- Entry in VALENTY found? -->
<xsl:when test="count($entries/VALENTY[@ATTR=string($longName)])!=0">
  <xsl:for-each select="$entries/VALENTY[@ATTR=string($longName)]">
    <xsl:if test="@FROM = $value">
      <xsl:value-of select="@TO"/>
    </xsl:if>
  </xsl:for-each>
  <xsl:value-of
select="$entries/VALENTY[@FROM=string($longName)]/@TO"/>
</xsl:when>
<!-- Entry in DELENTY found? -->
<xsl:when test="count($entries/DELENTY[@FROM=string($longName)])!=0">
  <!-- Do Nothing Set alias to null -->
</xsl:when>
<!-- ELSE -->
<xsl:otherwise>
  <xsl:value-of select="$value"/>
</xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:choose>
  <xsl:when test="$alias!=string('')">
    <xsl:element name="{ $name }">
      <xsl:value-of select="$alias"/>
    </xsl:element>
  </xsl:when>
  <xsl:otherwise>
    <xsl:copy-of select="."/>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

5.3 Weitere XSLT-Projekte mit INTERLIS/XML oder GML

Weitere uns zurzeit bekannte XSLT-Projekte mit INTERLIS/XML sind hier zusammengestellt:

- Diplomarbeit G. Schaad, Fachhochschule beider Basel (FHBB), Technischer Bericht, Departement Bau, Abt. VGI, Modul 6: "Modellierung der Abbildung INTERLIS-2 nach SVG". (g.schaad@geonova.ch).
- A. Morf und H.G. Gnägi, "Generating SVG From System Independent Conceptually Modeled Geodata", SVG Open / Carto.net Developers Conference, Zurich, Switzerland, July 2002. www.svgopen.org.

Uns zurzeit bekannte XSLT-Projekte mit GML:

- Generell: www.webmapping.org
- AmilGeo-Dateien, Uni der Bundeswehr, München, Prof. Wolfgang Reinhardt, 2000
- SVGOpen-Konferenz, Zürich, Juli 2002, www.svgopen.org
- GML und XSLT, www.ionicsoft.com
- GML und XSLT, www.galdosinc.com

Glossar

INTERLIS	<u>I</u> nter (zwischen) – <u>L</u> and – <u>I</u> nformations – <u>S</u> ysteme. Wenn in diesem Dokument von INTERLIS die Rede ist, wird immer von der Version 2.1 ausgegangen ([ILIREF2.1]).
UML	Unified Modeling Language. Eine grafische Darstellung von Modellen. Mehr siehe www.omg.org/uml/
GML	Geography Markup Language
XML	Extensible Markup Language (www.w3c.org)
XML Schema	Ersatz für DTD. Schema um den Aufbau eines XML-Dokumentes zu definieren (Datentypen, Einschränkungen etc.) ([XML_SCHEMA])
SQL	Abfragesprache für Datenbanken
OGC	Open Gis Consortium
W3C	World Wide Web Consortium. Homepage: www.w3c.org
INTERLIS/XML	XML Transfer für INTERLIS

Bibliographie

Referenzen:

- [GML_SPEC2.1.1] GML Implementation Specification. Zu finden auf der Open Gis Consortium Seite unter: <http://www.opengis.net/gml/02-009/GML2-11.pdf>
- [ILIREF2.1] INTERLIS Referenzhandbuch Version 2.1. Zu finden auf der INTERLIS Webseite unter: http://www.interlis.ch/docs_d.html#tag02
- [XML1.0] XML 1.0 Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/2000/REC-xml-20001006>
- [XML_NAMES] XML Namespace Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/REC-xml-names/>
- [XML_SCHEMA] XML Schema Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xmlschema-1/> und <http://www.w3.org/TR/xmlschema-2/>
- [XML_LINK1.0] XML Linking Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xlink/>
- [XML_POINTER1.0] XML Pointer Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xptr/>
- [XML_XSLT1.0] XML XSLT Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xslt/>
- [XML_XSLT2.0] XML Pointer Spezifikation (Working Draft). Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xslt20/>
- [XML_XPATH1.0] XPath Spezifikation. Zu finden auf der W3C Webseite unter: <http://www.w3.org/TR/xpath/>

Web-Links:

- Einführung in XSLT: <http://www.heise.de/ix/artikel/2001/01/167/>
- GML-Einführung: http://www.jlocationsservices.com/company/galdos/articles/GMLMapMaking_gml.htm

XSLT-Tutorials:	http://www.zvon.org/xxl/XSLTutorial/Output/example6_ch1.html
UML und XML Schema:	http://www.jrpit.flinders.edu.au/confpapers/CRPITV5Routledge.pdf
INTERLIS-Forum:	http://www.fhzh.ch/cgi-bin/wiki.pl?INTERLISForum/Ili2Vorteile
GML-Vortrag:	http://www.ikg.uni-bonn.de/Lehre/GIS_III/Folien/Pack-and-Go/unzip/gisIII.14.ppt
GML-Applikation:	http://www.ordnancesurvey.co.uk/business/
XSLT/XML-Kurzreferenzen:	http://www.mulberrytech.com/quickref/index.html
XSLT-Forum:	http://www.mulberrytech.com/xsl/xsl-list/index.html
XML in a Nutshell	ISBN: 3-89721-198-X bei O'Reilly
Essential XML	ISBN: 0-201-70914-7 bei Addison-Wesley