Hochschule Neubrandenburg
University of Applied Sciences

Masterarbeit

«Objektrelationales Mapping mit Anwendung im kvwmap»

zum
Erlangen des akademischen Grades
Master of Engineering
(M.Eng.)

Vorgelegt von: Matthias Pramme

1. Betreuer: Prof. Dr.-Ing. Andreas Wehrenpfennig
2. Betreuer: Dr.-Ing. Peter Korduan
Abgabedatum: 28.10.2011

urn:nbn:de:gbv:519-thesis2011-0070-1

Danksagung

An dieser Stelle möchte ich mich bei meinen beiden Betreuern,
Dr.-Ing. Peter Korduan und Prof. Dr.-Ing. Andreas Wehrenpfennig bedanken, die mich durch ihre hilfreichen Anregungen und ihre Geduld immer wieder unterstützt haben.

Nicht zuletzt möchte ich meiner Familie und meinen Freunden für ihre motivierende Unterstützung danken.

Kurzfassung

Das objektrelationale Mapping ist auch in der Geoinformatik eine aktuelle Problematik und behandelt die Abbildung von Objektstrukturen auf relationale Datenbanken. In dieser Thesis werden insbesondere Open-Source Anwendungen analysiert, die XML und GML in eine relationale Datenbank einlesen können. Für das WebGIS-Framework *kvwmap* wird ein Konzept aufgezeigt, das die Generierung von Karten-Layern aus dem Geodatenbanksystem PostGIS ermöglicht.

Schlagwörter: Objektrelationales Mapping, Datenbanken, XML, GML, GIS, kvwmap, PostgreSQL, PostGIS

Abstract

The object-relational mapping is also a current problem in geoinformatics and discusses the mapping of object structures to relational databases. In this thesis there's an analysis of open source software concerning to importing of XML and GML to relational databases. For the WebGIS framework *kvwmap* it is shown a concept that allows the generation of map layers from the spatial database system PostGIS.

Keywords: Object-relational mapping, database, XML, GML, GIS, kvwmap, PostgreSQL, PostGIS

Inhaltsverzeichnis

Glossar

AAA-Modell Anwendungsschema der Informationssysteme ALKIS, ATKIS und AFIS

API Application Programming Interface

Aggregation „Spezielle Form der Assoziation: Ganzes-Teil-Beziehung oder Hat-Ein-Beziehung“ [4].

Framework „Ist ein Umfeld von Anwendungen (Programmen), das auf bestimmte Bedürfnisse zugeschnitten werden kann“ [86].

Geoinformationssystem „Ein *Geoinformationssystem* (engl. *Geographic Information System, GIS*) stellt ein Informationssystem zur Erfassung, Speicherung, Verarbeitung und Darstellung von räumlichen Daten dar“ [8].

GUI Graphic User Interface: Graphische Oberfläche einer Anwendung

INSPIRE Infrastructure for Spatial Information in the European Community

JDBC Java Database Connectivity: Java-Schnittstelle für Datenbanksysteme

Komposition „Strenge Form der Aggregation, bei der die Teile vom Ganzen existenzabhängig sind“ [4].

NAS Normbasierte Austauschschnittstelle, Dateiformat des AAA-Modells

ODBC Open Database Connectivity, standardisierte Datenbankschnittstelle, die SQL als Anfragesprache benutzt.

OGC Open Geospatial Consortium

Referentielle Integrität „Referentielle Integrität sorgt für die Korrektheit und Konsistenz der Beziehungen zwischen den Relationen“ (vgl. [22]).

SHP Shapefile-Format: Ist ein von dem Unternehmen ESRI definiertes Format zur Verarbeitung von Vektor- und Rasterdaten.

W3C Das World Wide Web Consortium ist eine internationale Arbeitsgemeinschaft, die De-facto-Standards für World Wide Web-Technologien entwickelt.

XML „Extensible Markup Languange. XML ist eine ähnlich wie HTML aufgebaute Sprache, die Texte durch Auszeichnungs-Elemente hierarchisch gliedert.“ [44]

XML-Schema Die vom W3C spezifizierte Definitionssprache für XML.

XPlanGML Dateiformat des Vorhabens XPlanung in Deutschland

1. Einleitung

1.1. Motivation

Die Speicherung und der Austausch von Geodaten ist Thema vieler Projekte im Bereich der Geodäsie und Geoinformatik. Dazu zählen bspw. in Deutschland die GDI-DE, das AAA-Projekt [1] oder das Projekt XPlanung [34].

Speziell für den Austausch von Geodaten, insbesondere im Internet, hat sich das Datenformat GML (Geography Markup Language) in diesen genannten Projekten durchgesetzt, weil es u.a. eine verbesserte Interoperabilität zwischen Systemen ermöglichen und raumbezogene Daten (Geoobjekte) im lesbaren XML ausdrücken kann. Diese Spezifikation wurde von dem OGC (Open GIS Consortium) entwickelt und im Jahr 2007 zu der ISO-Norm 19136 erklärt (vgl. [29]). Die Definition von GML wurde durch Anwendung der XSD (XML Schema Definition) gelöst.

Im weiteren Sinne kann ein GML- oder XML-Dokument als Datenbank betrachtet werden, da es eine Sammlung von Daten beinhaltet (vgl. [41]). Trotzdem ist es eine Datei, die von Anwendungen zwar verarbeitet werden kann, aber nicht die Vorteile von Datenbankverwaltungssystemen erfahren kann. Darunter fallen insbesondere der Mehrbenutzerbetrieb, die Transaktionsverwaltung und die Datensicherung. Aus diesem Grund wird der Import von XML-Dokumenten in Datenbanken notwendig. Die Abbildung von XML-Strukturen oder XML-Daten auf Relationen ist Angelegenheit des objektrelationalen Mappings.

1.2. Aufgabenstellung

Geodaten für das Internet-GIS kvwmap werden zunehmend in XML geliefert, z.B. NAS oder xPlanung. Es ergibt sich das Problem die Daten in eine relationale Datenbank einzulesen. Bisher wurde für ein neues XML-Schema manuell ein Datenbankmodell erstellt, ein Importer implementiert und dann die Daten eingelesen. Um den Aufwand zu reduzieren und eine größere Flexibilität bei Änderungen der Schemata zu erreichen, soll dieser Prozess automatisiert werden. In einem ersten Schritt soll dabei aus dem Schema das Datenbankmodell abgeleitet, dann ein allgemeingültiger Importer entwickelt und schließlich die Definition der Layer für die GIS-Anwendung erzeugt werden.

Im Vorfeld der Arbeiten ist eine Recherche zur Überführung von XML-Schemata in relationale Datenbanken durchzuführen. Im Vordergrund sollen vorhandene Softwarelösungen aus dem OpenSource-Bereich stehen. Im Grundlagenteil soll die Überführung der Informationen aus einem XML-Schema in eine relationale Datenbank aufgezeigt werden. Der Fokus hierbei liegt auf datenzentrierte XML-Dokumente. Es ist aufzuzeigen welche Möglichkeiten es gibt, die Objektstrukturen, die im XML-Schema beschrieben sind in relationale Strukturen zu überführen. Bei der Umsetzung von Objekten mit Geometrie sollen die Datenstrukturen von PostGIS berücksichtigt werden. Insbesondere ist auch zu untersuchen in wie weit Entscheidungen bei der Überführung vom XML-Schema in das DB-Modell über einen aus dem Schema abgeleiteten Dialog abgefragt werden können, z.B. „Sollen Attribute eines Tags als Attribute einer Relation gespeichert werden?“

Folgende Teilaufgaben sind zu lösen:

- Erstellung eines allgemeinen Konzeptes für das Objektrelationale Mapping von XML Dateien in PostgreSQL-Datenbanken, den Import und die Erzeugung der Layerdefinitionen für kvwmap
- Implementierung des Datenbankschemagenerators
- Implementierung des Importers
- Implementierung des Layerdefinitionsgenerators

1.3. Aufbau der Arbeit

Die Arbeit ist in nachfolgende sechs Kapitel aufgeteilt. Der Grundlagenteil beschäftigt sich mit den Themen, die das objektrelationale Mapping betreffen. Dazu zählen relationale und objektrelationale Datenbanksysteme, sowie in Bezug auf das objektrelationale Mapping von Geodaten die Geodatenbanksysteme. Weiterhin werden die Formate XML und GML näher beleuchtet und ihre Überführung in relationale und objektrelationale Datenbanken aufgezeigt. In dem Kapitel 3 werden zunächst die Anforderungen herausgestellt und der Stand der Technik in GIS-Applikationen hinsichtlich des objektrelationalen Mappings erläutert. Anschließend werden freie und proprietäre objektrelationale Mapping-Tools analysiert. Im Kapitel 4 erfolgt eine Erläuterung der Anwendungsfälle und des Konzeptes für einen generischen Ansatz zum Abbilden von XML-Strukturen in relationalen Datenbanken. Außerdem wird ein Konzept für den Layerdefinitionsgenerator im WebGIS *kvwmap* vorgestellt. Das Kapitel 5 zeigt eine eigene prototypische Implementierung zur Erzeugung von Datenbankstrukturen aus dem W3C XML-Schema. Im Anschluss daran werden die Auswertungen zu Tests von einigen der in Kapitel 3 genannten Tools aufgezeigt. Die Ergebnisse werden in Kapitel 6 dargestellt und diskutiert. Anschließend erfolgt im Kapitel 7 eine Zusammenfassung und ein Ausblick.

2. Grundlagen

Dieses Kapitel beschäftigt sich mit den wesentlichen Grundlagen zu dieser Arbeit. Darunter fallen die Begriffe Datenbanken, XML, GML und GIS. Für ausgiebige Erläuterungen dieser Themenkomplexe wurden zahlreiche Bücher geschrieben. Aus diesem Grund können hier nur die wichtigsten Informationen vermittelt werden. Zunächst werden die einzelnen Datenbanksysteme behandelt, d.h. relationale Datenbanksysteme, objektrelationale Datenbanksysteme und Geodatenbanksysteme. Im Anschluss daran werden Technologien wie XML und GML vorgestellt und der Themenkomplex objektrelationales Mapping abgedeckt.

2.1. Datenbanksysteme

Ohne Datenbanken wird heutzutage kaum eine größere Software mehr betrieben (vgl. [63]). Die Vorreiter sind relationale, objektrelationale und objektorientierte Datenbanksysteme (s. Abschn. 2.3 Abb. 2.7).

2.1.1. Aufbau und Zweck

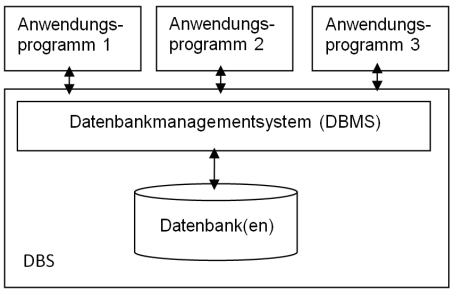


Abbildung 2.1.: Prinzipieller Aufbau: Datenbanksystem(vgl. [8])

Ein Datenbanksystem (DBS) besteht aus den Datenbanken und dem Datenbankmanagementsystem (DBMS (s. Abb. 2.1). Unter einem Datenbankmanagementsystem (DBMS) versteht man die Software, welche die Datenbanken verwaltet. Sie repräsentiert die Schnittstelle zwischen den Anwendern und den Datenbanken um für einen einheitlichen und effizienten Zugriff zu sorgen.

Die Vorteile von Datenbanken in einem DBMS gegenüber der dateibasierten Verwaltung liegen auf der Hand:

1. Sicherheit
2. Zuverlässigkeit und Konsistenzerhaltung durch Integritätsbedingungen
3. Persistenz und Effizienz
4. Verwaltung großer Datenmengen

Die Sicherheit, d.h. der Schutz der Daten vor unautorisierten Zugriffen, kann durch ein Rechtesystem gewährleistet werden. Somit können Zugriffsrechte für Benutzer angelegt werden, sodass versucht wird Datenverluste oder unbefugte Datenänderungen zu vermeiden. Eine weitere Eigenschaft von Datenbanksystemen betrifft die Zuverlässigkeit und Konsistenzerhaltung. In IT-Katastrophensituationen, wie z.B. Hardwarefehlern oder Stromausfällen, dürfen keine Daten verloren gehen. Die Datenbanken dürfen nicht in einen inkonsistenten Zustand überführt werden. Zur Konsistenzerhaltung wird das ACID-Transaktionskonzept angewandt (vgl. Abschn. 2.1.2). Ein Datenbanksystem kann durchaus an Effizienz gewinnen, wenn die Daten in einer Baumstruktur (z.B. R-Baum) verwaltet werden. Dadurch kann u.a. eine schnelle Such- oder Einfüge-Operation ermöglicht werden. Ein weiterer Vorteil von Datenbanksystemen ist die Verwaltung von großen Datenmengen, d.h. Datenmengen, die das Fassungsvermögen des Hauptspeichers übersteigen.

2.1.2. Transaktionskonzept

Unter einer Transaktion wird im Sinne der Datenbankentheorie eine Folge von Aktionen (INSERT, UPDATE, DELETE (vgl. Abschn. 2.2.2)) verstanden, über die eine Datenbank manipuliert wird (vgl. [16]). Das Transaktionskonzept (ACID-Prinzip, vgl. Tabelle 2.1) gilt für Datenbankmanagementsysteme und sieht vor, Inkonsistenzen zu vermeiden, indem die Integritätsbedingungen von Datenbanken eingehalten werden. Ein typisches Beispiel für eine Verletzung der Integrität wäre ein neuer Datensatz mit schon vorhandenem Primärschlüsselwert.

Atomarität (**A**tomicity)	Eine Transaktion wird entweder ganz oder gar nicht ausgeführt.
Konsistenz (**C**onsistency)	Transaktionen überführen eine Datenbank von einem konsistenten Zustand in einen konsistenten Zustand, d.h. Inkonsistenzen sind verboten.
Isolation (**I**solation)	Transaktionen laufen so ab, als würden sie alleine laufen, parallele Transaktionen haben keinen Einfluss auf das Ergebnis.
Dauerhaftigkeit (**D**urability)	Änderungen durch abgeschlossene Transaktionen bleiben erhalten, auch bei Systemausfall.

Tabelle 2.1.: ACID-Prinzip vgl. [17]

2.2. Relationale Datenbanksysteme

2.2.1. Aufbau von Relationen

Datenbanken können aus Tabellen bzw. Relationen aufgebaut werden. Diese werden als relationale Datenbanken bezeichnet. Eine Relation ist zweidimensional und besteht aus einer Anzahl von Zeilen und Spalten. Jede Spalte steht für ein Attribut (Merkmal bzw. Eigenschaft), das über einen Datentyp definiert wird. Hierbei werden die genormten Datentypen (vgl. ISO/IEC 9075:2008) wie z.B. *INT*, *FLOAT* oder *VARCHAR* benutzt. Die Datensätze liegen in Form von Zeilen bzw. Tupeln vor und können durch die Attribute beschrieben werden (s. Abb. 2.2). Ein Attribut kann als Schlüssel definiert werden, d.h. als Primär- oder Fremdschlüssel. Ein Primärschlüssel ist nicht notwendiger Weise für die Definition einer Relation erforderlich, wird aber dennoch unbedingt empfohlen, da dieser einen Datensatz identifiziert und somit von den anderen unterscheidet (vgl. [10]).

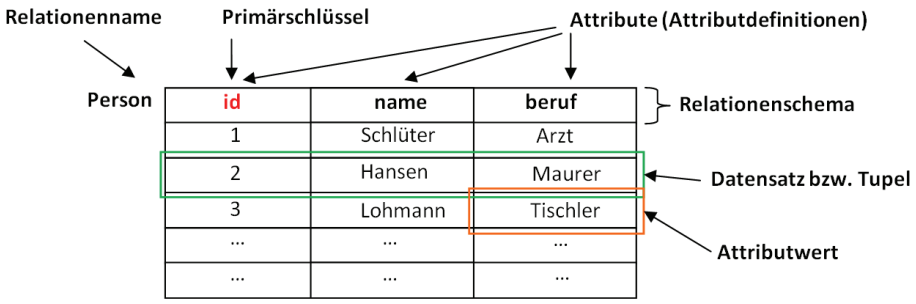


Abbildung 2.2.: Aufbau von Relationen

2.2.2. Datendefinition und Datenmanipulation

Die Data Definition Language (DDL) ist eine SQL-Sprache zur Erzeugung von Datenbankstrukturen.

Hauptbefehle sind *CREATE TABLE* (Tabelle erstellen), *ALTER TABLE* (Tabelle verändern) und *DROP TABLE* (Tabelle löschen).

Beispiel: Kundentabelle

```
CREATE TABLE kunde (
       id INTEGER,
       vorname VARCHAR(255),
       nachname VARCHAR(255),
       PRIMARY KEY (id)
);
```

Die Data Manipulation Language (DML) ist eine SQL-Sprache zur Veränderung der Daten einer Datenbank.

Befehle der DML sind *INSERT*, *UPDATE* und *DELETE* zum Einfügen, Ändern und Löschen von Daten.

Beispiel: Einfügen eines Kundendatensatzes

```
INSERT INTO kunde(id,vorname,nachname)
VALUES(1,'Joachim','Schumacher');
```

Beispiel: Änderung des Kundennamens nach Eheschließung

```
UPDATE kunde SET nachname='Schulz' WHERE nachname='Schumacher'
AND vorname='Joachim';
```

2.2.3. Datenabfrage

Die Data Query Language (DQL) ist eine SQL-Sprache zur Abfrage von Daten einer Datenbank. Zur Abfrage von Tabellen gibt es vier grundlegende Möglichkeiten:

- Abfrage des gesamten Datenbestandes einer Tabelle
  Beispiel: *SELECT * FROM kunde;*
- Selektion (engl. selection)
  Beispiel: *SELECT * FROM kunde WHERE nachname='Schulz ';*
- Projektion (engl. projection)
  Beispiel: *SELECT nachname, vorname FROM kunde;*
- Vereinigung (engl. union, Beispiel: *SELECT * FROM Mitarbeiter UNION SELECT * FROM Kunden* (vgl. [57])

Die Selektion bedeutet, dass eine Teilmenge aller Zeilen abgefragt wird, indem Bedingungen der Abfrage hinzugefügt werden. Die Projektion bildet das Pendant zur Selektion, indem hierbei eine Teilmenge aller Spalten ausgewählt wird. Bei der Vereinigung wird eine Abfrage über zwei oder mehrere Tabellen ausgeführt, wobei die Attribute den selben Datentyp besitzen müssen. Im Folgenden werden weitere komplexere Abfrage-Operationen aufgeführt:

- Schnittmenge (engl. intersection)
- Verbund (engl. join)
- Differenz (engl. minus)
- Quotient (engl. division)

2.2.4. Beziehungen

Zwischen Relationen können Beziehungen konzipiert werden. Drei verschiedene Beziehungen können im *Entity-Relation-Modell* auftreten:

- **1:1** → Ein Datensatz der Relation B wird einem Datensatz der Relation A zugeordnet und umgekehrt.

  Beispiel: „Ein zugelassenes Fahrzeug hat genau ein KFZ-Kennzeichen und jedes KFZ-Kennzeichen gehört zu genau einem Fahrzeug (Gilt nur für Deutschland. Die Schweiz z. B. hat Wechselkennzeichen.)“ ([79], s. Abb. 2.3).

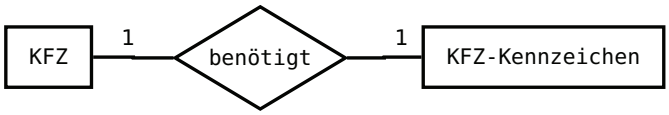


Abbildung 2.3.: ER-Modell: KFZ - KFZ-Zeichen

- **1:N** → Einem Datensatz der Relation A werden beliebig viele Datensätze der Relation B zugewiesen, ein Datensatz der Relation B kann aber nur einem Datensatz der Relation A angehören.

  Beispiel: Ein Gebäude kann aus mehreren Räumen bestehen, jeder Raum befindet sich aber in einem Gebäude (s. Abb. 2.4).

Abbildung 2.4.: ER-Modell: Gebäude - Räume

- **M:N** → Beliebig viele Datensätze der Relation A gehören zu beliebig vielen Datensätzen der Relation B und umgekehrt.

  Beispiel: Ein Buch kann von mehreren Autoren verfasst werden, jeder Autor kann wiederum mehrere Bücher schreiben (s. Abb. 2.5).

Abbildung 2.5.: ER-Modell: Bücher - Autoren

Die Realisierung erfolgt über Primär- und Fremdschlüssel. Der Primärschlüssel (engl. primary key, PK) sorgt in einer Relation für die Identifizierung eines Datensatzes. Dabei werden üblicherweise ganze Zahlen oder auch Zeichenketten verwendet, sofern sie dadurch von den anderen eindeutig unterschieden werden können. Der Fremdschlüssel (engl. foreign key, FK) ist die Referenz auf den Primärschlüssel und enthält denselben Schlüsselwert. Im Sinne der geforderten Datenbankkonsistenz (vgl. [67]) kann die referentielle Integrität, d.h. die Sicherstellung ordentlicher Beziehungen über das jeweilige Datenbankmanagementsystem überprüft werden. Ein typisches Beispiel für die Verletzung der referentiellen Integrität ist das Einfügen eines Datensatzes mit schon vorhandenem Primärschlüssel.

Bei 1:1-Beziehungen (Beipiel: KFZ - KFZ-Zeichen) können die Attribute der Klassen auch in einer Relation zusammengefasst werden. Im anderen Fall gibt es zwei Optionen die Relationen miteinander zu verbinden. Eine Option sieht vor für beide Relationen dieselben Primärschlüssel zu verwenden, wobei einer zum Fremdschlüssel wird. Die andere besteht darin, einen Fremdschlüssel in einer der Relationen zu definieren. Relationen, die eine 1:N-Beziehungen aufweisen, werden über Primärschlüssel und Fremdschlüssel miteinander verbunden. Der Fremdschlüssel muss in der Relation mit der Kardinalität N definiert werden.

Bei M:N-Beziehungen wird eine *Zwischentabelle* bzw. *Beziehungstabelle* eingeführt, in der die Fremdschlüssel mit Verweis auf die Primärschlüssel der beteiligten Relationen definiert werden (s. Abb. 2.6).

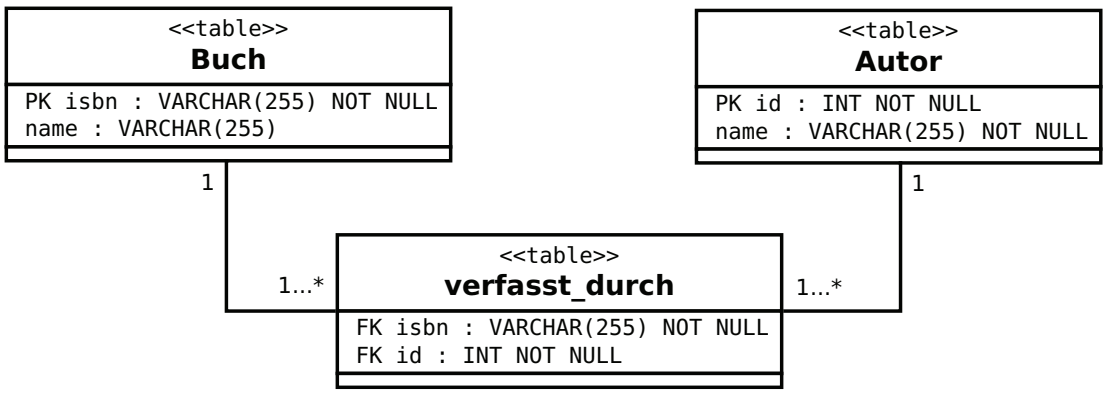


Abbildung 2.6.: Datenbankmodell: Bücher - Autoren

2.3. Objektrelationale Datenbanksysteme

Nach Aufkommen der objektorientierten Programmiersprachen (Smalltalk (70er Jahre), C++ (70er bis 80er Jahre), Java (90er Jahre), etc. (vgl. [26])) wurde eine Reihe von neu konzipierten objektorientierten Datenbanksystemen entwickelt (vgl. [8], S.18). Seitdem hatten sich relationale Datenbanksysteme auf dem Markt bewährt. Die Unternehmen wollten diese beibehalten, d.h. nicht unbedingt die neuen Technologien nutzen, weil ihre Anwendungen zumeist auf relationale Datenbanken zugeschnitten waren und ein Wechsel auch einen erheblichen Aufwand an Schnittstellen-Programmierung zur Folge gehabt hätte (vgl. [8], S.18). Die stärkere Nutzung von relationalen Datenbanken zeigt auch die Prognose des Unternehmens IDC[1] für den Zeitraum 1999-2004 bei Analyse der Marktanteile von Datenbanksystemen (s. Abb. 2.7).

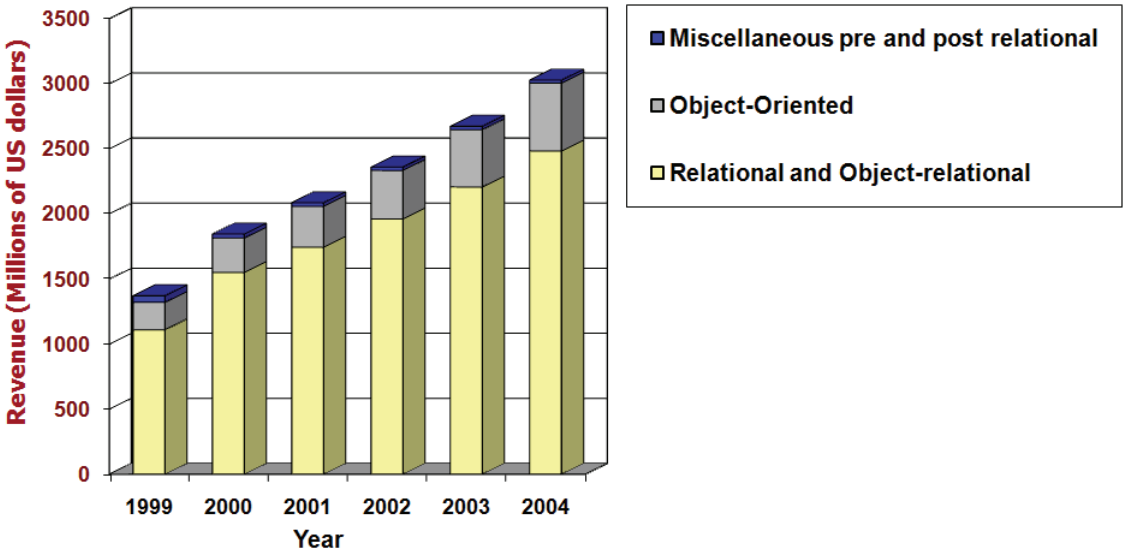


Abbildung 2.7.: Prognose: Marktanteile der Datenbanksysteme von 1999-2004 [9]

Deswegen wurde mit den objektrelationalen Datenbanken eine Lösung geschaffen, die den Einsatz von relationalen Datenbanken mit Konzepten der Objektorientierung ermöglicht. Somit bleiben die ursprünglichen Eigenschaften von relationalen Datenbanken erhalten, die Verarbeitung von Tabellen mit einigen Objekteigenschaften ist aber zusätzlich realisierbar. Eine Relation kann als Klasse von Objekten (Datensätze bzw. Tupel) mit gleichen Eigenschaften verstanden werden.

[1] IDC (International Data Corporation): Internationales Marktforschungs- und Beratungsunternehmen in den Bereichen Informationstechnologie und Telekommunikation (vgl. [28])

2.3.1. Prinzipien der objektorientierten Programmierung

In objektrelationalen Datenbanksystemen wird das relationale Modell durch Prinzipien aus der objektorientierten Programmierung (OOP) erweitert. In der Praxis können aber meist nur objektorientierte Datenbanksysteme, wie z.B. Oracle, das gesamte Spektrum der Objektorientierung enthalten (vgl. [18]).

Zu den Prinzipien zählen:

- Abstraktion: Klassen und Objekte
- Kapselung
- Polymorphie
- Vererbung und Mehrfachvererbung
- Persistenz

Jedes Phänomen der realen Welt kann als Objekt einer Klasse abstrahiert werden. Eine Klasse entspricht einem komplexen Datentyp, wobei dieser aber zusätzlich Methoden enthalten kann. Das Prinzip der Kapselung aus der objektorientierten Programmierung sieht vor, den Zugriff auf die Daten (Objekteigenschaften) nur über ihre Schnittstelle (Objektverhalten) zu gestatten (vgl. [66]). Somit kann der Zustand eines Objektes von außen ohne „Einwilligung“ des Objektes nicht verändert werden und ein konsistenter Zustand erreicht werden. Das wird bspw. in der Programmiersprache Java durch Deklaration der Instanzvariablen mit den Zugriffsmodifikatoren *protected* (sichtbar für eigene und abgeleitete Klassen) oder *private* (sichtbar für eigene Klasse) gewährleistet. Unter der Polymorphie wird im Allgemeinen die „Vielgestaltigkeit“ eines Objektes verstanden. Dahinter verbergen sich verschiedene Konzepte, wie z.B. das Überladen von Operatoren und Methoden oder die Typkonvertierung. Mittels der Vererbung (Generalisierung und Spezialisierung) können Klassenhierarchien ermöglicht werden, wodurch eine Klasse die Schnittstelle und evt. die Eigenschaften seiner Elternklasse erbt. Wenn eine Klasse mehr als eine Elternklasse hat, wird das Konzept als Mehrfachvererbung bezeichnet. Das Konzept kann zu Unübersichtlichkeiten und Namenskonflikten führen (Diamond-Problem, vgl. [60]) und wird deswegen von vielen Programmiersprachen (z.B. Smalltalk, Java und C#) nicht unterstützt (vgl. [58]). Die Realisierung kann aber trotzdem z.B. über Schnittstellen (Interfaces) erfolgen.

Die Persistenz von Objekten stellt ein weiteres Prinzip der objektorientierten Programmierung dar. Um Objekte (Daten) dauerhaft zu erhalten bzw. zu persistieren, müssen diese auf einem Datenträger oder in einer Datenbank gespeichert (Serialisierung) und bei Verwendung von einer Applikation wieder geladen (Deserialisierung) werden können (vgl. [40]).

2.3.2. Unterschiede zu relationalen Datenbanken

Im Gegensatz zu den relationalen Datenbanken wird die erste Normalform, d.h. die Bedingung, dass in einer Relation nur atomare Attribute zugelassen werden, aufgehoben. Somit sind zusammengesetzte bzw. komplexe Datentypen (Objekttypen) erlaubt. Außerdem wird die Möglichkeit geschaffen eigene Operationen und Funktionen über Programmiersprachen zu definieren (vgl. [8], S. 23). Dadurch gestaltet sich ein Datenbanksystem flexibler und der Funktionsumfang kann erheblich erweitert werden.

Einige objektrelationale Datenbankmanagementsysteme wie bspw. IBM DB2 unterstützen die Vererbung der Objekttypen und der Klassen (Relation) (vgl. [85]).

Der Vorteil gegenüber relationaler Datenbanken ist die Unterstützung von Programmiersprachen. Außerdem zeichnen sich objektrelationale Datenbanken durch Einführung von speziellen Objektidentitäten (OID) aus.

2.3.3. PostgreSQL

PostgreSQL ist ein quelloffenes objektrelationales Datenbankmanagementsystem, das auf fast allen Plattformen eingesetzt werden kann und der heutigen SQL-Norm (ISO/IEC 9075:2008) gerecht wird (vgl. [52]). Es wird vermehrt auch in geowissenschaftlichen Anwendungen (Mapbender, deegree und QGIS) genutzt, weil es durch das PlugIn *PostGIS* auch Geodaten verarbeiten kann (s. Abschn. 2.4.2). Eine typenbasierte Vererbung wird nicht unterstützt, nur eine tabellenbasierte Vererbung (s. Listing. 2.1).

Die Anpassung von PostgreSQL-Datenbanken kann durch Verwendung geeigneter Bibliotheken über prozedurale (C,Perl) und objektorientierte Programmiersprachen (Java, Python usw.) erfolgen (vgl. [42], [50]).

```
CREATE TABLE stadt (
    name                VARCHAR(255),
    gruendungsdatum     DATE,
    bevoelkerung        FLOAT,
    bevoelkerungsdichte FLOAT,
    ...
);

CREATE TABLE land (
   id SERIAL Primary Key,
   name VARCHAR(255),
   kuerzel VARCHAR(255),
   ...
);

CREATE TABLE hauptstadt (
   land_id   INT,
   CONSTRAINT fk_hauptstadt_land
     FOREIGN KEY(land_id) REFERENCES land(id)
) INHERITS (staedte);
```

Listing 2.1: Modellierung von Hauptstädten

2.4. Geodatenbanksysteme

Geodatenbanksysteme kommen überall dort zum Einsatz, wo Daten einen Raumbezug haben. Das beginnt beim einfachen Gebäudeinformationssystem und endet beim komplexen Rauminformationssystem, wie z.B. GeoScopeAVS (vgl. [43]). Der Fachausdruck Geodatenbanksystem oder auch räumliches Datenbanksystem wird verwendet, wenn die Datenbanken die Speicherung von Geodaten und die Bearbeitung von räumlichen Anfragen unterstützen (vgl. [8]). In dem Datenbanksystem PostgreSQL können Geodatenbanken mittels der Erweiterung PostGIS erstellt werden.

2.4.1. Datenbanken mit Raumbezug

Die Notwendigkeit von Geodatenbanksystemen wird deutlich, wenn die Verwaltung von Geodaten in relationalen Datenbanken betrachtet wird. Angenommen die Bundesländer der Bundesrepublik Deutschland sollen für eine relationale Datenbank modelliert werden. Die Geometrie eines Bundeslandes kann über ein MultiPolygon definiert werden. Es handelt sich dabei um eine Anzahl von Flächen bzw. Polygone. Jedes Polygon kann Aussparungen enthalten und daher mehrere Ringe besitzen, die sich aus mindestens drei Punkten zusammensetzen. Der Sachverhalt kann wie in Abb. 2.8 gestaltet werden.

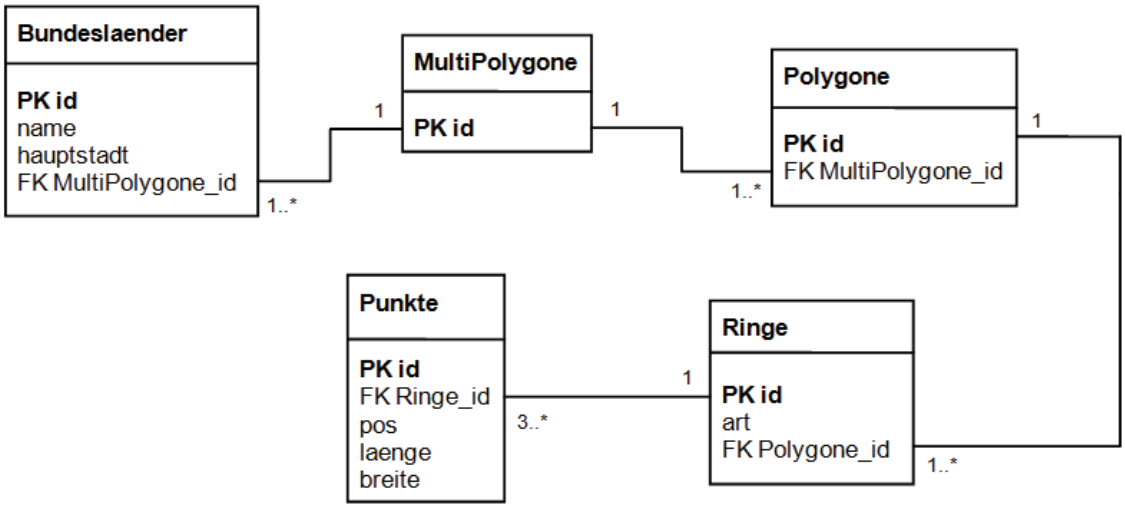


Abbildung 2.8.: Beziehungsdiagramm für Multipolygone (vgl. [8], S. 26)

```
SELECT Punkte.laenge, Punkte.breite, Ringe.id, Ringe.art
     FROM Bundeslaender b, MultiPolygone m, Polygone poly, Ringe r, Punkte p
      WHERE b.name = 'Schleswig-Holstein' AND
            b.MultiPolygone_id = m.id AND
            m.id = poly.MultiPolygone_id AND
            poly.id = r.Polygone_id AND
            r.id = p.Ringe_id
      ORDER BY m.id, poly.id, r.id, p.pos;
```

Listing 2.2: SQL-Abfrage MultiPolygone

Nach Erstellung des Datenbankschemas wird einem erfahrenen Datenbankbenutzer schnell das Ausmaß der resultierenden Anwendungsschwierigkeiten vor Augen geführt. Die Abfrage erweist sich als kompliziert und unflexibel (s. Listing 2.2). Die Modellierung von komplizierteren Geometrien würde noch mehr Relationen enthalten und dadurch sehr unübersichtlich werden. Hinzu kommt, dass das Datenbankschema wie in der Abb. 2.8 auch u.a. noch kein Referenzsystem und keine Dimension enthält.

Objektrelationale Datenbanken eignen sich viel eher für die Verwaltung von Geodaten, weil sie geometrische Datentypen sowie geometrische und topologische Funktionen zur Verfügung stellen. Somit wäre die Lösung für das Beispiel eine Tabelle *Bundeslaender* mit den Attributen *id*, *name*, *hauptstadt* und einer Geometriespalte von dem Datentyp *MultiPolygon*. In Postgis ist ein solcher Datentyp schon vorhanden.

2.4.2. PostgreSQL/PostGIS

Das objektrelationale Datenbanksystem *PostgreSQL* wird nach Installation der Erweiterung *PostGIS* fähig Geodaten zu verwalten. *PostGIS* stellt Geometrietypen, Koordinatenreferenzsysteme, räumliche Operatoren und Funktionen für Geodaten bereit. Somit können PostgreSQL-Datenbanken zu Geodatenbanken umfunktioniert werden (vgl. [53]). PostGIS richtet sich nach dem OGC-Standard *OpenGIS Simple Features Specification for SQL* bzw. ISO-Norm 19125-1:2004 und der ISO-Norm SQL/MM Spatial (ISO/IEC 13249-3:2006). Das Simple Feature Modell (s. Anhang A) ist eine Kollektion aus dem Spatial Schema (ISO-Norm 19107:2003) für zwei-dimensionale Geodaten (vgl. [38]). Geometrien der dritten und vierten Dimension stehen aber auch durch die Extended Well-Known-Datentypen zur Verfügung.

Eine Datenbank wird in PostgreSQL „PostGIS-tauglich“, nach dem die Operatoren und Funktionen (SQL-Skripte der PostGIS-Installation lwpostgis.sql und spatial_ref_sys.sql) installiert worden sind, die prozedurale Sprache plpgsql aktiviert und die Tabellen geometry_columns und spatial_ref_sys generiert worden sind. Eine Geometriespalte wird in PostGIS über die Funktion *AddGeometryColumn* definiert. Zuvor muss eine Relation definiert worden sein. Als Parameter müssen mindestens der Relationenname, der Name der Geometriespalte, das Referenzsystem mittels EPSG-Code (-1 für jedes Referenzsystem), der Geometrietyp (Geometry, Point, Linestring, Polygon usw. (s. Anhang A)) und die Dimension angegeben werden. Eine Geometriespalte kann auch in der Tabellendefinition definiert werden, indem einem Attribut der Datentyp Geometry zugewiesen wird. Das hat aber den Nachteil, dass keine Geometrieüberprüfung (*CONSTRAINTS*, bzgl. Referenzsystem, Geometrie und Dimension) stattfinden kann.

Erstellung der Relation und Hinzufügen einer Geometriespalte im WGS 84:

```
CREATE TABLE strasse(
       id INT NOT NULL PRIMARY KEY, strname VARCHAR(255), ...
);

SELECT AddGeometryColumn('strasse', 'the_geom', 4326, 'LINESTRING', 2 );
```

2.5. XML

XML ist ein Akronym und steht für eXtensible Markup Language. Darunter wird eine Auszeichnungssprache bzw. Metasprache verstanden, die von dem World Wide Web Consortium (W3C) entwickelt wurde. Unter einem Markup versteht man eine Textauszeichnung, d.h. die Gestaltung von Text (z.B. kursive oder farbliche Schrift), aber auch die Kennzeichnung der Datenstruktur mittels Tags. „XML ist keine Programmiersprache, weil ihr Elemente für die Steuerung von Programmabläufen fehlen“ [68]. XML ist aus seinem Vorgänger SGML (Standard Generalized Markup Language) entstanden und kann als weiterentwickelte Teilmenge von SGML verstanden werden. SGML diente zur Repräsentierung von Dokumenten in digitaler Form. Durch seinen komplexen Kern wurde aber schnell klar, dass es für das Web weniger geeignet ist. Weiterhin ist XML im Vergleich zu SGML viel restriktiver, d.h. die Syntax ist strenger geregelt. Im Gegensatz zu SGML müssen bei XML u.a. Start- und End-Tags existieren und die Tags dürfen nicht leer sein (vgl. [11]). Eine Übersicht der wachsenden XML-Familie kann unter *http://kensall.com/big-picture/bigpix22.html* angesehen werden.

2.5.1. XML-Dokument

XML-Dokumente sind selbstbeschreibend, weil sie Daten über Daten enthalten (Metadaten). Zum einen können über die Tags Aussagen zum Inhalt gemacht werden und zum anderen können Attribute das Dokument näher beschreiben.

„Viele Publikationen zum Thema XML und Persistenz teilen XML-Dokumente in zwei Kategorien ein: dokumenten- und datenzentriert“ [45]. Dokumentenzentrierte XML-Dokumente sind solche, die sehr viel Text aufweisen und deren Struktur sehr variieren kann. Sie sind insbesondere zur Präsentation für den Menschen geeignet, ein Paradebeispiel sind die digitalen Bücher (Ebooks).

Datenzentrierte XML-Dokumente (s. Listing 2.3) sind hingegen für eine vielseitige Prozessierung vorgesehen. Sie enthalten vorrangig Daten und sind durch eine festgelegte, exakt definierte und gleichbleibende Struktur gekennzeichnet (vgl. [45]). In der Geoinformatik zählen dazu insbesondere die Anwendungsschemata (s. Abschn. 2.6.4).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <klasse xmlns="http://www.eine-schule.de/schule"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.eine-schule.de/schule schulklasse.xsd">
5
6  <lehrer id="1" name="jons" vorname="horst">
7    <schueler id="1" name="kon" vorname="jÃ¼rgen"></schueler>
8    <schueler id="2" name="schlÃ¼ter" vorname="hans"></schueler>
9    <schueler id="3" name="jack" vorname="gerd"></schueler>
10 </lehrer>
11 </klasse>
```

Listing 2.3: Datenzentriertes XML-Dokument schulklasse.xml

Wohlgeformtheit und Validität

Die Syntax von XML ist in gewissem Maße vorgeschrieben. Hierfür hat das W3C 89 Regeln verfasst (vgl. [73]).

Wohlgeformt ist ein XML-Dokument, wenn es u.a. folgende Bedingungen einhält (vgl. [76], [73]):

- XML-Dokument beginnt mit der XML-Deklaration (Prolog)
- XML-Dokument muss ein eindeutiges Root-Element besitzen
- Die Bezeichner der Start- und End-Tags müssen identisch sein (korrespondierende Tags)
- Elemente sind case-sensitiv (Groß- Kleinschreibung)
- Elemente müssen geschlossen werden und können aber auch leer sein (<elementname/>)
- Elemente müssen richtig geschachtelt werden, d.h. die Tags dürfen sich nicht überlagern (Fehler: <hotel><hotelnummer></hotel></hotelnummer>)
- Attribute müssen in Anführungszeichen gesetzt werden
- Entities müssen für spezielle Zeichen verwendet werden (z.B. & für &)

Valide XML-Instanzen müssen wohlgeformt sein und darüber hinaus von einer DTD oder einem XML-Schema abgeleitet sein und allen Deklarationen des Inhaltsmodells, die in der DTD oder dem XML-Schema definiert wurden, entsprechen. (vgl. [51]). XML-Schemata können auch auf Wohlgeformtheit oder Validität überprüft werden. Validierende Parser sind bspw. in Projekten wie XMLSpy oder Xerces und JAXP enthalten.

2.5.2. DTD

Die Dokumenttyp-Definition (DTD) dient zur Modellierung von XML-Dokumenten. Somit wird die Syntax in dem Sinne weiter eingeschränkt, wobei diese schon ohnehin durch die XML Regeln gewisser Maßen festgelegt ist.

```
<!ELEMENT Person (Name, Vorname, Adresse)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Adresse (Strasse,PLZ,Ort)+>
<!ELEMENT Strasse (#PCDATA)>
<!ELEMENT PLZ (#PCDATA)>
<!ELEMENT Ort (#PCDATA)>
```

Die DTD kann nicht über DOM oder SAX verarbeitet werden. Ein DTD-Parser ist von Nöten.

Weitere Probleme bzw. Defizite:

- Keine Unterstützung von bekannten Datentypen
- Keine Festlegung von Wertebereichen
- Keine Standardwerte nur Konstanten (#FIXED)
- Keine Kontrolle der Eingaben (z.B. reguläre Ausdrücke)
- Kein XML, Parser-Aufwand
- Keine Namensräume

2.5.3. XML-Schema

Eine Weiterentwicklung bzw. Technologie des W3C zur Lösung der DTD-Probleme stellt das XML-Schema dar. Das Ziel war aus den Einschränkungen der DTD zu lernen und diese zu beseitigen. XML-Schema ist selber XML, d.h. die XML-Syntax wird auch zur Definition von XML-Dokumenten benutzt. Dadurch ergibt sich ein geringerer Parser-Aufwand, weil der Parser zum Verarbeiten der XML-Dokumente auch XML-Schemata zergliedern kann. Am 15. Februar 1999 formulierte das W3C folgende Anforderungen für das XML-Schema (vgl. [68], [69]):

- Einführung von Namensräumen
- Vererbungsmechanismen (substitutionGroup, oder restriction), keine Mehrfachvererbung
- Einschränkung des Inhaltsmodells von Elementen
- Integration von weiteren XML-Schemata
- Berücksichtigung bekannter Datentypen aus Programmiersprachen (z.B. integer, string und byte)

Im XML-Schema wird definiert welche Elemente und Attribute in dem XML-Dokument auftreten dürfen. Dabei können sich diese auch in entfernten XML-Schemata befinden und über den Namensraum referenziert werden. Außerdem können die Bezeichner, Anzahl und Reihenfolge der Kindelemente festgelegt werden. Elemente können leer sein oder Text beinhalten. Für die Elemente und Attribute können Datentypen definiert werden. Es besteht auch die Möglichkeit Konstanten oder vorgegebene Werte für Elemente und Attribute zu definieren. Die dazugehörige Spezifikation ist die XML Schema Definition Language (XSD).

Datentypen

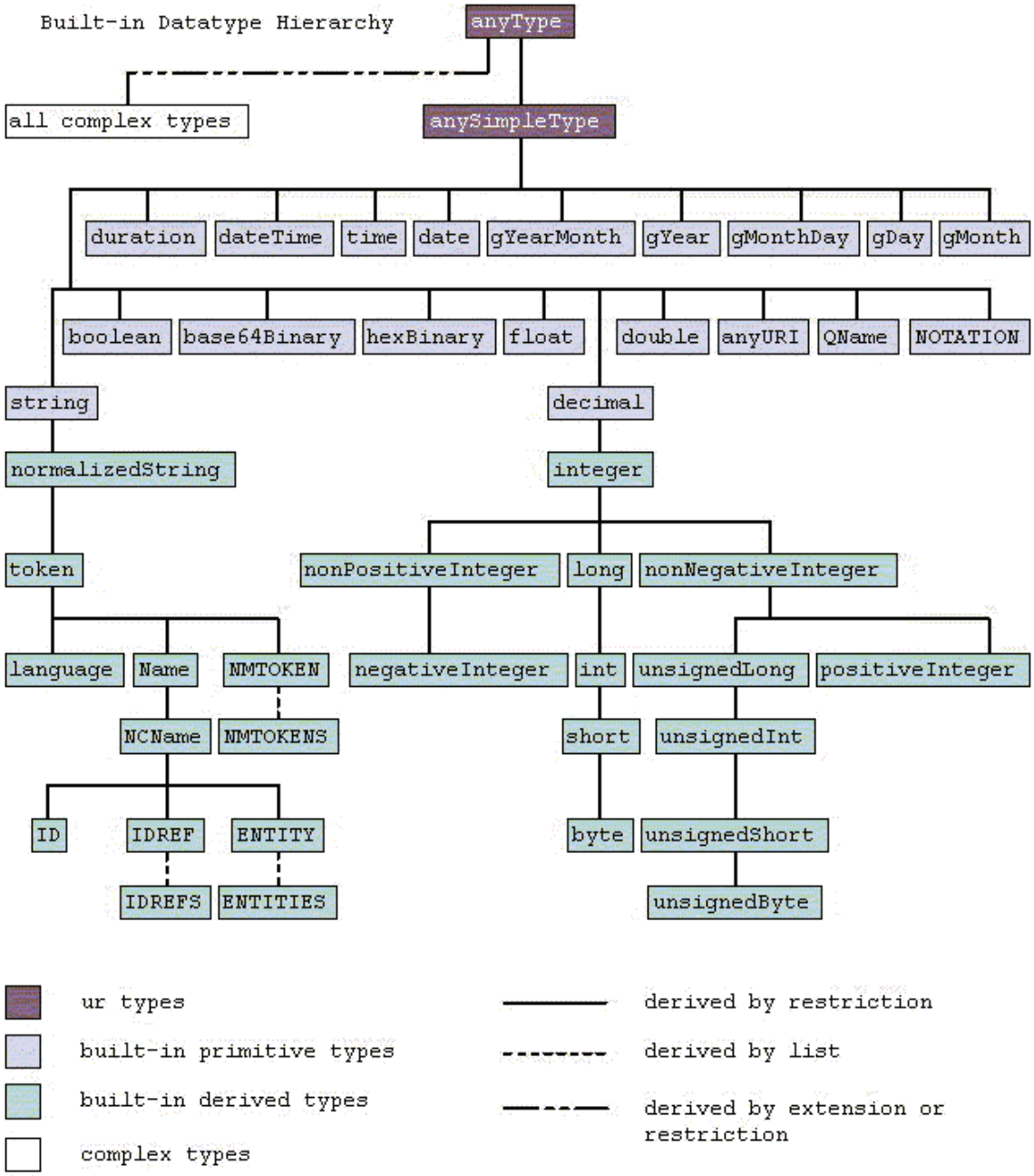


Abbildung 2.9.: Datentypen-Hierarchie der XSD [70]

Die Datentypen der XSD werden in die Obergruppen *String*, *Date*, *Numeric* und *Misc* (engl. miscellaneous für sonstige) eingeteilt. Die Datentypen der Gruppe *String* beinhalten u.a. die Repräsentation von Zeichen-, Zeichenketten und Whitespaces (z.B. Zeilenvorschub, Leerzeichen). Die Datentypen der Gruppe *Date* umfassen u.a. Zeit, Datum, Jahr, Monat und Tag (s. Abb. 2.9)). Für die Darstellung von Zahlen werden die bekannten Datentypen wie z.B. *integer*, *double* oder *float* verwendet. Für Dezimalzahlen sind weitere in der Hierarchie kleiner werdende Datentypen durch Anwendung von Restriktionen definiert worden (s. Listing 2.4).

```
<xs:simpleType name="short" id="short">
  <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-2/#short"/>
  </xs:annotation>
  <xs:restriction base="xs:int">
    <xs:minInclusive value="-32768" id="short.minInclusive"/>
    <xs:maxInclusive value="32767" id="short.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>
```

Listing 2.4: Definition des Datentypen *short*

Im XML-Schema können auch zusammengesetzte bzw. komplexe Datentypen definiert werden, wobei diese eine große „Schachtelungstiefe“ annehmen können (s. Listing 2.5).

```
<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Listing 2.5: Komplexer Typ [77]

Elemente

Der Aufbau des XML-Schema besteht aus einem Wurzel-Element und weiteren Unterelementen. Das Wurzel-Element-Tag muss nach dem Bezeichner *xs:schema* benannt sein (vgl. Listing 2.6), weil es aus dem Namespace *xmlns:xs='http://www.w3.org/2001/XMLSchema'* benutzt wird, damit ein XML-Schema auch als solches identifiziert werden kann.

Elemente können in einer *ModelGroup* deklariert werden zu Element-Folgen (<sequence>) und Element-Alternativen (<choice>). Weiterhin können Elemente auch von dem xml-Tag <all> umschlossen werden. Dann sind alle darin enthaltenen Elemente optional und daher auch alle Kombinationen von Element-Folgen möglich.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

Listing 2.6: Wurzel-Element

Einfache Elemente

Ein einfaches Element kann nur aus Textinhalten bestehen, d.h. es kann keine weiteren Elemente beinhalten. Die Textinhalte müssen nicht notwendigerweise vom Datentyp *string* sein. Alle Standard-Datentypen (string, integer, boolean, etc.) und eigene Datentypen sind erlaubt.

```
<xs:element name="name" type="xs:string"/>
<xs:element name="strasse" type="xs:string"/>
<xs:element name="plz" type="xs:integer"/>
<xs:element name="email" type="emailType"/>
<xs:simpleType name="emailType">
  ^.*[@].*[.].*$
</xs:simpleType>
```

Listing 2.7: Einfache Elemente

Komplexe Elemente

Komplexe Elemente können aus Textinhalten und Elementen bestehen. Die Elemente eines komplexen Elementes können wiederum eine komplexe Struktur ausweisen (s. Listing 2.8).

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>

The following schema declares the "letter" element:
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 2.8: Beispiel: Versandankündigung

Attribute

Attribute können nur in komplexen Typen vorkommen und sind per „default“ optional. Durch den Parameter *required* werden sie erst zur Notwendigkeit. Mehrere Attribute können auch in Attributgruppen zusammengefasst und referenziert werden.

2.5.4. Programmierschnittstellen für XML

Im folgenden Abschnitt werden abstrakte Programmierschnittstellen vorgestellt. Diese sind abstrakt, weil sie keine Programmiersprachen abhängige Implementierungen darstellen, sondern generische Techniken für den Zugriff auf XML-Dateien bereitstellen (vgl. [68], S. 355).

DOM

Das Document Object Model ist eine Empfehlung des W3C und definiert eine Programmierschnittstelle, die XML-Dateien als Knotenbaum betrachtet und entsprechend zergliedern kann für weitere Analysen oder Manipulationen. Das Knotenmodell wird im Arbeitsspeicher vorgehalten und kann jederzeit während eines Prozesses abgerufen werden. Aufgrund des hierarchischen Modells können Verwandtschaften zwischen den einzelnen Knoten hergestellt werden und somit die Knoten über entsprechende Methoden der API angesprochen werden. Das XML-Listing 2.9 stellt das Instanzdokument einer Bücherei dar. In der Abbildung 2.10 ist das entsprechende DOM zu sehen.

```
<!-- Edited by XMLSpy® -->
<bookstore>
 <book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
 </book>

  <book category="children">
   <title lang="en">Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
  </book>

  <book category="web">
   <title lang="en">XQuery Kick Start</title>
   <author>James McGovern</author>
   <author>Per Bothner</author>
   <author>Kurt Cagle</author>
   <author>James Linn</author>
   <author>Vaidyanathan Nagarajan</author>
   <year>2003</year>
   <price>49.99</price>
  </book>

  <book category="web" cover="paperback">
   <title lang="en">Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
  </book>
 </bookstore>
```

Listing 2.9: Books.xml [75]

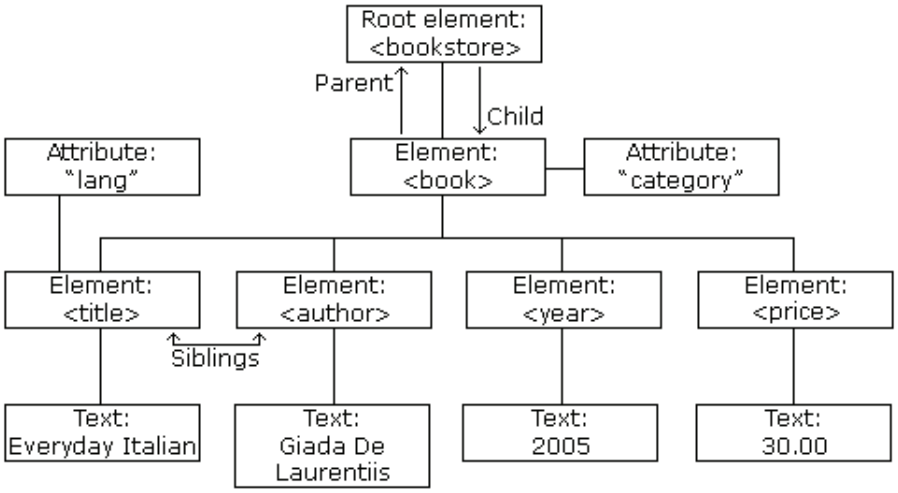


Abbildung 2.10.: Knotenbaum, [74]

SAX

Im Gegensatz zu DOM ist die Simple API for XML (SAX) ein Ereignis-basiertes Verfahren zur Verarbeitung von XML. Nachteilig ist, dass der Zugriff auf die XML-Komponenten nur während des Parser-Vorgangs erfolgen kann, weil kein Objektmodell gespeichert wird. Der Vorteil ist aber die Verarbeitung von großen Dokumenten, bei denen DOM an Effizienz verliert (vgl. [65]).

2.5.5. XSL

XSL ist ein Kurzwort und steht für den englischen Begriff eXtensible Stylesheet Language und bezeichnet eine vom W3C entwickelte XML-basierte Stylesheet-Sprache. Denkt man zunächst an Stylesheet-Sprachen wird dem einen oder anderen sofort CSS (Cascading Style Sheets) einfallen, welches benutzt wird um HTML-Seiten zu formatieren. XSL ist nicht nur eine Stylesheet-Sprache, sondern besteht aus den drei Komponenten:

- XSLT (XSL Transformations) : Zur Umwandlung von XML-Dokumenten
- XPath (XML Path Language) : Zur Navigation in XML-Dokumenten
- XSL-FO (eXtensible Stylesheet Language Formatting Objects) : Zur Formatierung von XML-Dokumenten

2.6. GML

2.6.1. Allgemeines

Die Geography Markup Language (GML) stützt sich auf die XML-Familie und erweitert das XML-Schema um weitere Typen (Geometrie und Topologie) explizit für Daten mit Raumbezug. Sie ist eine Spezifikation des Open Geospatial Consortium (OGC) und fungiert als eine Metasprache für Geoinformationen.

GML wurde im Jahr 2007 zur ISO-Norm 19136 (vgl. [29]) erklärt und im Jahr 2009 vom Deutschen Institut für Normung (DIN) übernommen (vgl. [14]).

Normen und Standards, die GML verwendet, sind:

- W3C-Standards
  - XML 1.0
  - XML Schema (Parts 0, 1, 2)
  - XML namespaces
  - XPointer/XPath
  - XLink
- ISO-Normen
  - ISO TC/211 (19103, 19107, 19108, 19109, 19111, 19112, 19117, 19123)

Im Zusammenhang mit Geoinformationssystemen (GIS) wird dieses Format im Zuge der Normung von vielen Anwendungen (z.B. ArcGIS, MapServer, deegree) unterstützt und auch der Versuch unternommen dieses in relationalen Datenbanken zu verwalten (s. Abschn. 3.2).

2.6.2. Aufbau

Das GML-Modell ist modular aufgebaut. Während zu Beginn der Entwicklung in der Version 1.0.0 noch das Modell über die DTD definiert wurde und nur Geoobjekte[2] mit einfachen (graphischen) Geometrien (Punkte, Linien, Polygone und Aggregate von diesen) verfügbar waren, so wird es in der derzeitigen Version 3.2.1 vom XML-Schema getragen und enthält deutlich mehr Eigenschaften zur computergestützten Verarbeitung (s. Abschn. 2.6.3). Die Abb. 2.11 zeigt die Klassenhierarchie der aktuellen GML Version 3.2.1. Deutlich zu erkennen ist die modulare Struktur, die eingeteilt ist in wesentliche Merkmale von Geoinformationen. Zu nennen sind vor allem die Geoobjekte (abgeleitet von *gml:AbstractFeature*), ihre Geometrie (*GML Geometry*), ihre Topologie[3] (*GML Topology*)

[2] Geoobjekt (engl. geographic feature): „abstraction of real world phenomena“ [30].

[3] Die Topologie beschäftigt sich mit den nichträumlichen und strukturellen Beziehungen beliebiger Elemente in abstrakten Räumen (Definition nach Bill:1999).

und die Klasse für die Koordinatenreferenzsysteme (*GML Coordinate Reference System*). Die Klasse *GML Metadata* befindet sich zwar noch im GML-Schema. Sie wurde aber von dem OGC abgelehnt.

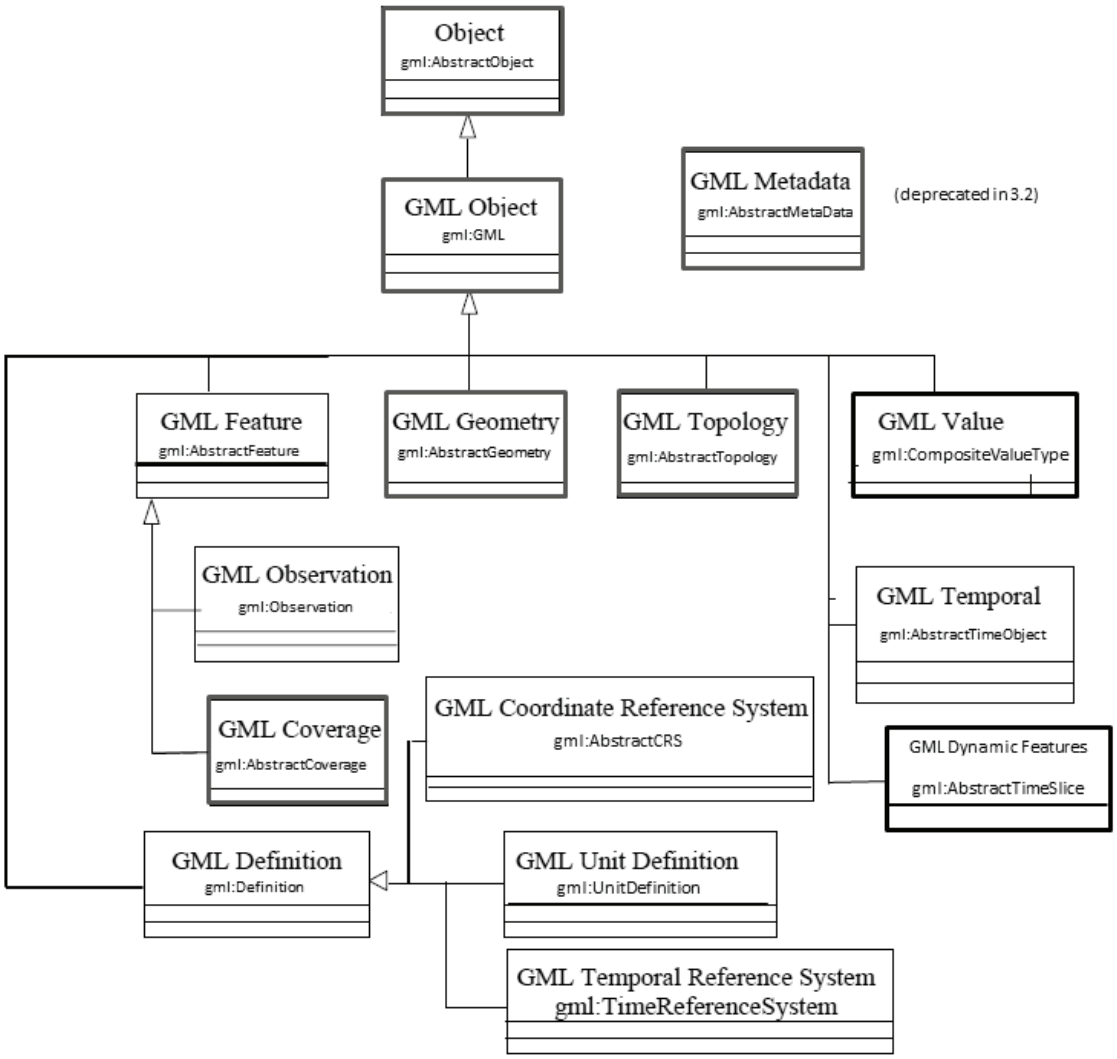


Abbildung 2.11.: GML 3.2.1 Klassenhierarchie, erstellt von Ron Lake (Galdos Systems Inc.)

Die Abb. 2.12 zeigt das Klassenmodell einer Stadt. Die Stadt ist ein Geoobjekt und erbt deswegen die Eigenschaften des Features. Sie beherbergt mehrere Straßen, wobei jede Straße notwendiger Weise ein Linienzug (LineString) ist.

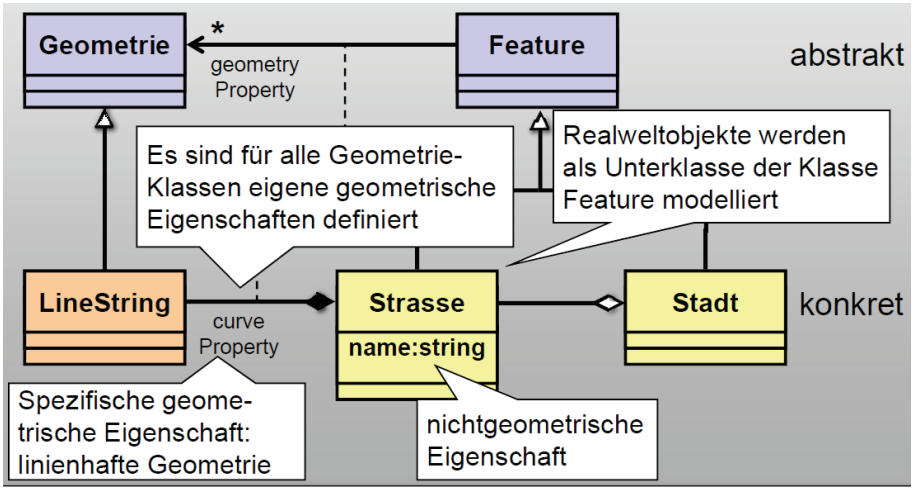


Abbildung 2.12.: Beispiel: Straße [36]

Das Listing 2.10 ist die Implementierung des Klassenmodells (Abb. 2.12).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.beispiel.
       de/stadt" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xlink="http://www.
       w3.org/1999/xlink" targetNamespace="http://www.beispiel.de/stadt"
       elementFormDefault="qualified">
3    <xs:import namespace="http://www.opengis.net/gml/3.2" schemaLocation="
         gmlProfilexplan.xsd"/>
4    <xs:element name="Stadt" type="StadtType" substitutionGroup="gml:AbstractFeature
         "/>
5    <xs:complexType name="StadtType">
6      <xs:complexContent>
7        <xs:extension base="gml:AbstractFeatureType">
8          <xs:sequence>
9            <xs:element name="Strasse" type="StrasseType" maxOccurs="unbounded"/>
10         </xs:sequence>
11       </xs:extension>
12     </xs:complexContent>
13   </xs:complexType>
14   <xs:complexType name="StrasseType">
15     <xs:complexContent>
16       <xs:extension base="gml:AbstractFeatureType">
17         <xs:sequence>
18           <xs:element ref="gml:LineString" maxOccurs="unbounded"/>
19         </xs:sequence>
20       </xs:extension>
21     </xs:complexContent>
22   </xs:complexType>
23 </xs:schema>
```

Listing 2.10: GML-Schema stadt-beispiel.xsd

Ein korrespondierendes GML-Dokument ist in Listing 2.11 dargestellt. Es handelt sich um die Robert-Blum-Straße der Stadt Neubrandenburg im Gauss Krüger-System (4. Streifen Krassowski-Ellipsoid, 3 Grad).

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <Stadt xmlns="http://www.beispiel.de/stadt" gml:id="stadt1"
3   xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xlink="http://www.w3.org/1999/
       xlink"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.beispiel.de/stadt stadt-beispiel.xsd">
6  <gml:name>Stadtmodell NB</gml:name>
7  <gml:boundedBy>
8    <gml:Envelope srsDimension="2" srsName="http://www.opengis.net/def/crs/EPSG
       /0/2398">
9      <gml:lowerCorner>4575032.95 5925671.99</gml:lowerCorner>
10     <gml:upperCorner>4591881.64 5942075.93</gml:upperCorner>
11   </gml:Envelope>
12 </gml:boundedBy>
13 <Strasse gml:id="strasse1">
14 <gml:name>Robert-Blum-Strasse</gml:name>
15  <gml:LineString gml:id="strzug1">
16   <gml:posList>4582895.57 5937943.12 4582943.92 5937960.16 4583142.55 5937989.73
       4583296.65 5938006.42</gml:posList>
17  </gml:LineString>
18 </Strasse>
19 </Stadt>
```

Listing 2.11: GML-Instanzdokument stadt-beispiel.xml

2.6.3. Unterschiede zwischen GML2 und GML3

Die Tabelle 2.2 verdeutlicht die Unterschiede zwischen den Versionen 2 und 3 der GML-Spezifikationen. Darunter fallen die Änderung des Identifikators eines Geoobjektes und die in der Version 2 fehlende Kodierung der Topologie. Abstrakte Elemente werden in der derzeitigen Version 3.2.1 nicht mehr durch einen Unterstrich, sondern durch Hinzufügen des Adjektives *Abstract* gekennzeichnet. Außerdem enthält die Version 3 im Gegensatz zur Version 2 weitere Geometrien, wie z.B. Gitternetze, Splines oder Polyederflächen.

	GML 2	**GML 3.0**	**GML 3.1**	**GML 3.2**
Objekt-ID	fid, optional		gml:id, optional	gml:id, required
Kennz. abstrakter Elemente	Unterstrich (Beispiel: _Feature)			explizite Schreibweise (Beispiel: AbstractFeature)
Geometrie Erweiterungen u.a.	Punkte, Linien, Polygone und Aggregate von diesen	Orientierte Kurve, Splines, Komplexe bestehend aus Kurve, Fläche oder Körper	Polyederfläche, Gitternetze (Zylinderfläche, Kegelfläche, Sphere), Klothoide und TIN	gml:Envelope ist keine Geometrie, gml:Shell (Außenhülle eines Körpers) (vgl. [48])
Topologie	nein	Knoten, Kante, Masche, Körper, Orientierung		
Weitere	nein	Kodierung von zeitlichen Abläufen, Definition von eigenen Referenzsystemen, sowie Maßeinheiten und Datenqualitätsinformationen (vgl. [64])		

Tabelle 2.2.: GML-Versionskonflikte

Das OGC plant die 4. Version von GML und eröffnete zu diesem Vorhaben am 19. September 2011 einen Workshop um jede Idee mit einfließen zu lassen (vgl. [21]). Einige Ideen tendieren dazu GML für das Semantic Web fit zu machen, in dem xlink durch rdf ausgetauscht wird. Andere Mitglieder sprechen sich für die Vereinfachung der Datenstrukturen aus.

„One primary example where complexity may be reduced is the encoding of geometries, for example it should be possible to encode a Point in just a single element like <Point x=.. y=.. [srs=EPSG:N..] /> i.s.o. multiple nested elements that do not add semantics“ (Just van den Broecke, [21]).

2.6.4. GML-Anwendungsschema

Anwendungsschema (engl. application schema): „conceptual schema for data required by one or more applications“ [30].

Ein GML-Anwendungsschema ist in XML-Schema verfasst und besteht aus einem GML-Profil und anwendungsspezifischen Definitionen (vgl. [38]). Es umfasst die detaillierte Beschreibung der Datenstrukturen und -inhalte von Anwendungen (vgl. [38], [31]) und soll in der Sprache UML (Unified Modeling Language) modelliert werden (vgl. [31]). Außerdem soll es unter Einbehaltung der *Regeln für Anwendungsschemata (engl. Rules for application schema, ISO 19109:2005)* erstellt werden. Diese Regeln sind zur Erreichung von Interoperabilität und damit weltweiten Einsatzes unbedingt notwendig (vgl. [38]). Jeder GML-Parser soll dadurch in der Lage sein jedes Anwendungsschema zu verarbeiten. Die Regeln für Anwendungsschemata schreiben die Verwendung des General Feature Models (GFM) vor. Das GFM ist ein Metamodell, in dem definiert wird, nach welchen Vorgaben ein Geoobjekt modelliert werden sollte. In der Abb. 2.13 ist der Weg von der Modellierung eines Sachverhalts der Realitätswelt hin zu einem Anwendungsschema zu sehen.

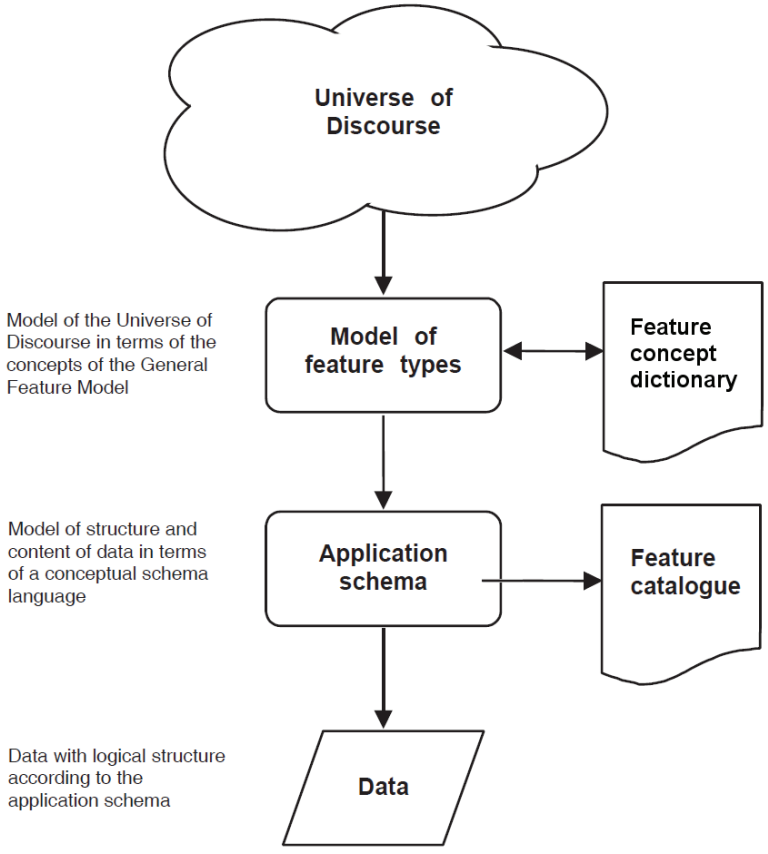


Abbildung 2.13.: From reality to geographic information [31]

Eine Liste von GML-Anwendungsschemata ist unter dem Link *http://www.ogcnetwork.net/node/210* zusammengestellt worden.

Die Abbildung 2.14 zeigt einen Auszug aus dem GML-Profil der Anwendungsschemata AAA und Xplanung. Es werden nicht alle Geometrien benötigt, die GML unterstützt.

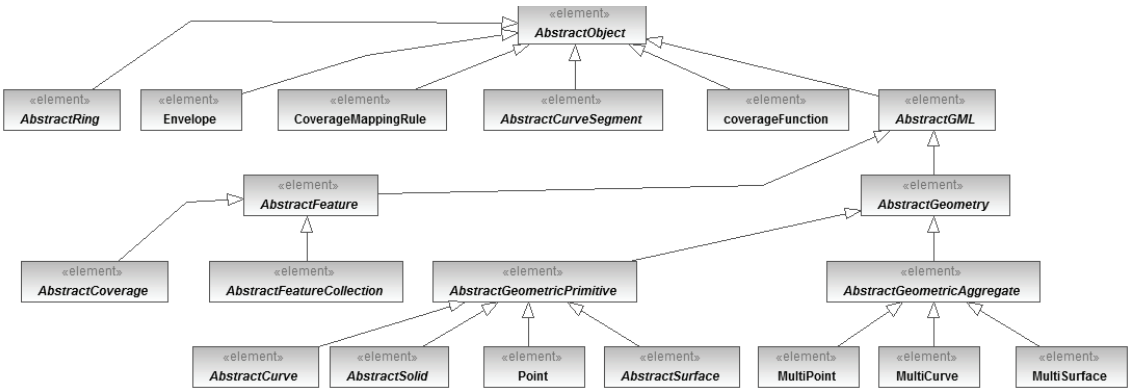


Abbildung 2.14.: Auszug des GML-Profils 3.2.1 der Formate NAS 6.0 und XPlanGML 4.0

2.7. Objektrelationales Mapping

Unter der objektrelationalen Abbildung oder dem objektrelationalen Mapping (kurz O/R-Mapping oder ORM) versteht man im Allgemeinen Techniken um ein Objekt-Modell auf ein relationales Datenmodell abzubilden oder umgekehrt (vgl. [25], S.xi).

Bezogen auf XML sind das Techniken um die DTD oder das XML-Schema in eine objektrelationale Datenbank zu übertragen oder aus einem Datenbankschema zu erzeugen. Hierzu kann die DTD oder das XML-Schema zunächst in ein Objekt-Modell und anschließend in ein objektrelationales Modell umgewandelt werden, wobei diese beiden Schritte zu einem direkten Datenbank-Mapping kombiniert werden können. Ronald Bourret ist der Ansicht, dass der Begriff objektrelationale Abbildung eher unzutreffend ist, weil dies nicht dem kompletten Anwendungsspektrum entspricht. Er erachtet den Begriff objektbasierte Abbildung eher für sinnvoll (vgl. [5]). Die aus der XML-Struktur gewonnenen Objekte können nicht nur in objektrelationale Datenbanken exportiert werden, vielmehr sind nichtrelationale, objektorientierte oder hierarchische Datenbanken denkbar (vgl. [5]). Trotzdem wird hier bei dem Begriff objektrelationales Mapping verblieben, weil dieser zumeist verwendet wird.

2.7.1. Objektrelationale Unverträglichkeit

Die Problematik, die das O/R-Mapping lösen soll wird als *objektrelationale Unverträglichkeit* (engl. object-relational impedance mismatch) bezeichnet (vgl. [2]).

Die *JBoss Comunity* drückt sich zu diesem Begriff folgender Maßen aus:

„Object-Relational Impedence Mismatch' (sometimes called the 'paradigm mismatch') is just a fancy way of saying that object models and relational models do not work very well together“ [23].

Die Tabelle 2.3 zeigt die Unterschiede zwischen dem objektorientierten und dem relationalen Modell. Bezogen auf die strukturellen Unterschiede bestehen die Daten im objektorientierten Modell aus Objekten, die Daten und Methoden enthalten können. Im relationalen Modell liegen aber nur Daten in Tabellen vor (s. Abb. 2.2). Nun ist in der Objektorientierung aber auch die Vererbung oder Kapselung (Verbergen) von Daten möglich, wobei diese Konzepte vom relationalen Modell nicht unterstützt werden. Daher müssen Vererbungshierarchien im relationalen Modell z.B. durch Fremdschlüsselbeziehungen gelöst werden. In Bezug auf die Identität von Objekten oder Tupeln gibt es auch wesentliche Unterschiede. In objektorientierten Programmiersprachen gibt es zwei Konzepte um die Identität von Objekten zu überprüfen, die Objektidentität und die Objektgleichheit. Eine Objektidentität[4] liegt vor, wenn sich bspw. zwei Objekte auf dieselbe Adresse im Speicher beziehen. Objekte sind hingegen objektgleich[5], wenn diese dieselben Objektmerkmale (Attribute, Konstruktoren, Methoden, usw.) besitzen.

Ein weiterer Unterschied bezieht sich auf die Navigation im jeweiligen Modell. Für eine Abfrage des Objektmodells wird der Objektbaum durchlaufen. Im relationalen Modell muss aber dem Abfragemechanismus die Datenstruktur vorgegeben werden, die abgefragt werden soll, meistens über *JOIN-Abfragen*.

	Objektorientiertes Modell	**Relationales Modell**
Struktur	Daten (Eigenschaften) und Methoden (Verhalten), objektorientierte Konzepte (Vererbung, Kapselung)	Daten (Attribute)
Identität	Objektidentität und Objektgleichheit	Primärschlüssel
Beziehungen	Assoziation, Aggregation und Komposition	Schlüsselkonzept
Daten-Navigation	Objektmodell durchlaufen	Navigation muss bekannt sein, mittels JOIN können mehrere Tabellen verbunden werden

Tabelle 2.3.: Unterschiede in den Modellen

[4]Objektidentität: Überprüfung in Java mittels == Operator
[5]Objektgleich: Überprüfung in Java mittels *equals - Funktion*

2.7.2. Mapping-Verfahren

Kanonische Abbildung

Die kanonische Abbildung bedeutet, dass das Objektmodell in nahezu ähnlicher Weise auf die relationale Datenbank abgebildet werden soll. Somit taucht jede 1:1 und 1:N Beziehung im relationalen Modell wieder auf. Wenn das Objektmodell eine sehr komplexe Struktur aufweist, kann das zu zahlreichen Tabellen führen.

Generische Tabellenstruktur

Ein Konzept, das allgemein gültig für das Speichern von Objekten in relationalen Datenbanken sein kann, ist in der Abb. 2.15 dargestellt. Die Vererbung wird über eine Relation *Inheritance* realisiert, in der die *id* der Super- und Subklasse einer jeden Klasse gespeichert wird. Die Attribute einer Klasse werden in der Relation *Attribute* und der entsprechende Attributwert und Attributtyp in den Relationen *Value* und *AttributeType* abgelegt. Das Verfahren kann bei gering verzweigten Objekten sehr effizient sein (vgl. [49]), aber eignet sich bspw. nicht für komplexe XML-Schemata.

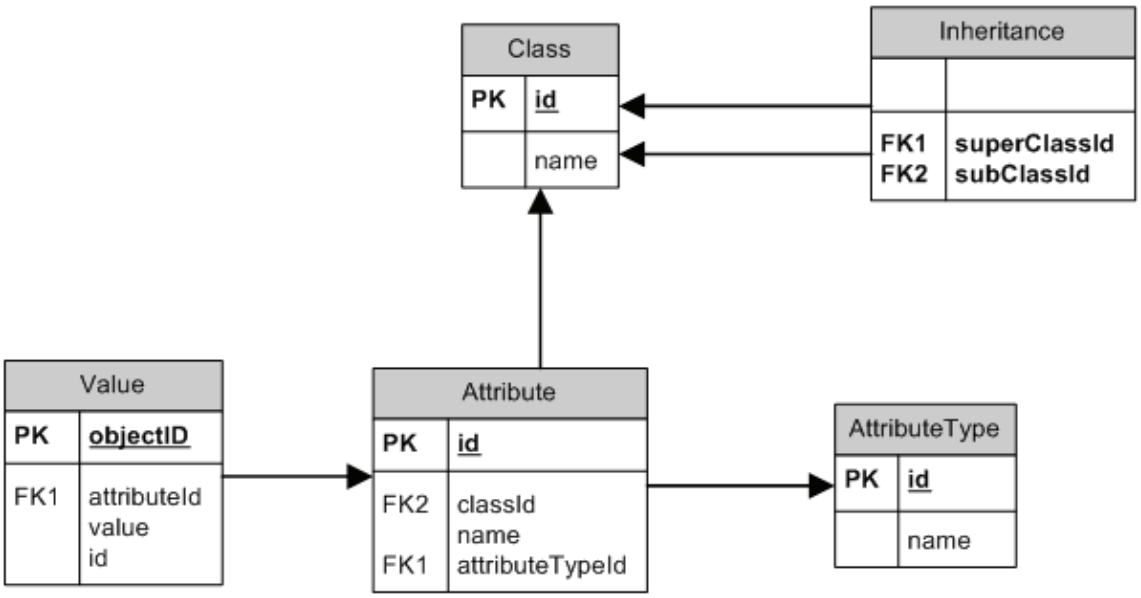


Abbildung 2.15.: Generische Tabellenstruktur (vgl. [49])

Abbildung der Vererbung

Zur Abbildung der Kind-Eltern-Beziehung von Objekten werden drei Strategien unterschieden (vgl. [12]):

- Eine Tabelle pro Subklasse: Subklasse referenziert die Superklasse
- Eine Tabelle pro Klassenhierarchie: Eigenschaften der Subklassen werden in Superklasse eingebaut
- Eine Tabelle pro konkreter Klasse: Eigenschaften der Superklassen werden in Subklassen eingebaut

Das Listing 2.12 zeigt die Java-Implementierung des Modell eines Schülers (Pupil). Ein Schüler ist eine Person und erbt deswegen alle Eigenschaften der Klasse Person.

```java
public class Person{
  private String lastname = null;
  private String firstname = null;

  public Person(String lastname,
                String firstname) {
    this.lastname = lastname;
    this.firstname = firstname;
  }

  public Person() {}

  public void setLastname(String lastname){
    this.lastname = lastname;
  }

  public String getLastname(){
    return this.lastname;
  }

  public void setFirstname(String firstname){
    this.firstname = firstname;
  }

  public String getFirstname(){
    return this.firstname;
  }
}
```

```java
public class Pupil extends Person{
  private int schoolyear = -1;

  public Pupil(String lastname,
               String firstname,
               int schoolyear){
    super(lastname, firstname);
    this.schoolyear = schoolyear;
  }

  public Pupil(){}

  public void setSchoolyear
              (int enrollment){
    this.schoolyear = schoolyear;
  }

  public int getSchoolyear(){
    return this.schoolyear;
  }

  public void printPupil(){
    System.out.println(
    "<- Object Pupil ->"
    +"\n\tlast name: "
    +getLastname()
    +"\n\tfirst name: "
    +getFirstname()
    +"\n\tschool year: "
    +getSchoolyear());
  }
}
```

Listing 2.12: Java-Klassen Person und Pupil

Das Diagramm 2.16 auf der nächsten Seite, zeigt ein potentielles Modell für die Abbildung der Java-Klassen nach der Strategie „Eine Tabelle pro Subklasse“. Hier wird aber deutlich

das die Methoden der Java-Klassen nicht in der relationalen Datenbank abgebildet werden können.

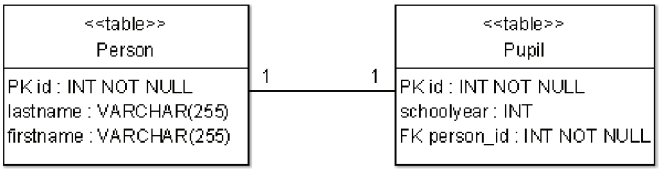


Abbildung 2.16.: Datenbankmodell des Beispiels Person und Pupil

2.7.3. Mapping der DTD

Die Document Type Definition (DTD) dient zur Festlegung einer XML-Datenstruktur. Somit werden Einschränkungen für die Wahl eines Datenmodells getroffen um die Validität eines XML-Dokuments prüfen zu können. Für den Export von XML-Dateien in Datenbanken ist die Umwandlung der XML-Datenstruktur in ein Datenbankmodell relevant. Dabei können nach Bourret (vgl. [5]) zwei Verfahren zur Abbildung unterschieden werden, die tabellenbasierte (table-based Mapping) und die objektrelationale Abbildung (O/R-Mapping). Beide Verfahren haben die Eigenschaft, dass sie bidirektional sind, d.h. die originale XML-Datenstruktur kann auch aus dem erzeugten Datenbankmodell rekonstruiert werden.

Tabellenbasierte Abbildung

Bei der tabellenbasierten Abbildung wird das XML-Dokument als eine oder mehrere Tabellen betrachtet. Aus dem XML-Dokument werden daher eine oder mehrere Tabellen generiert. Die XML-Dokumente müssen eine der in Listing 2.13 veranschaulichten Strukturen aufweisen.

```
<Table>                          <Tables>
  <Row>                            <Table_1>
    <Column_1>...</Column_1>         <Row>
    ...                                <Column_1>...</Column_1>
    <Column_n>...</Column_n>           ...
  </Row>                               <Column_n>...</Column_n>
  ...                              </Row>
  <Row>                            ...
    <Column_1>...</Column_1>       </Table_1>
    ...                            ...
    <Column_n>...</Column_n>       <Table_n>
  </Row>                             <Row>
</Table>                               <Column_1>...</Column_1>
                                       ...
                                       <Column_m>...</Column_m>
                                   </Row>
                                 </Table_n>
                               </Tables>
```

Listing 2.13: XML-Strukturen der tabellenbasierten Abbildung (vgl. [5])

In Abb. 2.17 wird ein einfaches Beispiel aufgeführt, das eine tabellenbasierte Abbildung einer Raumverwaltung zeigt. Ein Gebäude besteht aus mehreren Räumen, wobei jeder Raum eine Anzahl von IT-Komponenten (PCs und Drucker) hat.

```
<gebaeude>
  <raum>
    <id>1</id>
    <pc>30</pc>
    <drucker_anz>2</drucker_anz>
  </raum>
  <raum>
    <id>2</id>
    <pc>25</pc>
    <drucker_anz>2</drucker_anz>
  </raum>
</gebaeude>
```

==>

Tabelle gebaeude

id	pc	drucker_anz
1	30	2
2	25	2

Abbildung 2.17.: Tabellenbasierte Abbildung

Das Verfahren ist einfach und kann bei kleinen Datenmengen durchaus effizient sein. Hier wird aber deutlich, dass nur kleine einfache XML-Dokumente benutzt werden sollten, weil bei großen Datenmengen die Attributanzahl der Relationen erheblich zunimmt und ohne Normalisierung Redundanzen verbleiben. Ein weiterer Nachteil ist der Verlust an Informationen, weil die physikalische Struktur (Referenzen, CDATA-Abschnitte, Zeichenkodierung, standalone-Deklaration), Dokumentinformation, Kommentare und Verarbeitungsanweisungen nicht erhalten bleiben (vgl. [5]). Aus diesen Gründen wird das Verfahren eher auf dem umgekehrten Wege in XML-fähigen Datenbanken (z.B. PostgreSQL oder Oracle) genutzt um aus den Datenbanktabellen XML-Dokumente zu generieren.

Objektrelationale Abbildung

Die Abb. 2.18 zeigt das Prinzip des O/R-Mappings anhand eines Beispiels. Hier handelt es sich um den Verkauf von Artikeln, bzw. die Assoziation Angebot - Artikel. Üblicherweise können mehrere Artikel in unterschiedlichen Angeboten auftauchen (N:M-Beziehung), aber um das Beispiel einfach zu gestalten wird hier eine 1:N-Beziehung angenommen, d.h. ein Angebot kann aus mehreren Artikeln bestehen und jeder Artikel ist genau einem Angebot zugeordnet.

DTD

```
<!ELEMENT Angebot (Preis,Artikel+)>
<!ATTLIST Angebot id CDATA #REQUIRED>
<!ELEMENT Artikel (Name,Hersteller,Preis)>
<!ATTLIST Artikel id CDATA #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Hersteller (#PCDATA)>
<!ELEMENT Preise (#PCDATA)>
```

Klassen	**Tabellen**
Klasse Angebot {	Tabelle Angebot {
int id;	Attribut id INTEGER (PK)
int preis;	Attribut preis VARCHAR
Artikel[ ] a;	}
}	
Klasse Artikel {	Tabelle Artikel {
int id;	Attribut id INTEGER (PK)
String name;	Attribut name VARCHAR
String hersteller;	Attribut hersteller VARCHAR
int preis;	Attribut preis VARCHAR
}	Attribut angebot_id INTEGER (FK)
	}

Abbildung 2.18.: Verkauf von Artikeln, O/R-Mapping der DTD

Die Abbildung eines korrespondierenden XML-Dokuments wird in Abb. 2.19 dargestellt.

XML-Dokument

```
<Angebot id='100'>
  <Preis>3.59</Preis>
  <Artikel id='1'>
    <Name>Bleistift</Name>
    <Hersteller>Schumann</Hersteller>
    <Preis>1</Preis>
  </Artikel>
  <Artikel id='2'>
    <Name>Collegeblock</Name>
    <Hersteller>mpaper</Hersteller>
    <Preis>2.59</Preis>
  </Artikel>
</Angebot>
```

==>

Objekte

```
Objekt Angebot {
  id = 100
  preis = '3.59'
  Artikel = Ref. zu Artikeln
}

Objekt Artikel {
  id = 1
  name = 'Bleistift'
  hersteller = 'Schumann'
  preis = '1'
}

Objekt Artikel {
  id = 2
  Name = 'Collageblock'
  hersteller = 'mpaper'
  preis = '2.59'
}
```

Tabellen

Tabelle Angebot

id	preis
100	3.59

Tabelle Artikel

id	id_angebot	name	hersteller	preis
1	100	Bleistift	Schumann	1
2	100	Collegeblock	mpaper	2.59

Abbildung 2.19.: Verkauf von Artikeln, O/R-Mapping des XML-Dokuments

Grundkonzept

Die DTD wird von einem Parser gelesen und in seine Strukturen zerlegt. Elemente werden zu Klassen, einfache Kindelemente werden zu Attribute, Referenzen werden zu Fremdschlüsseln, ids der Elemente zu Primärschlüsseln.

DTD-Komponente	**Datenbankschema**
Element mit Kind-Elementen (A((B,C)|D))	Relation
Sequenz von Elementen (A,B,C)	Attribute einer Relation
Alternative von Elementen (A|B|C)	Attribute einer Relation, Nullwerte sind erlaubt
Kind-Element mit der Kardinalität ? (optional, d.h. 0 oder 1)	Attribute einer Relation, Nullwert ist erlaubt
Kind-Element mit der Kardinalität + (mind. 1 oder N) oder * (0 oder N)	Attribut als Liste bzw. Array oder Relation mit Beziehung zum Elternelement, wobei Nullwerte bei * erlaubt sind
Attribut	Attribut einer Relation
Attribut mit der Option *#IMPLIED*	Attribut einer Relation, Nullwert ist erlaubt
Attribut mit der Option *#REQUIRED*	Attribut einer Relation, Attributwert ist erforderlich, Nullwert ist nicht erlaubt
Attribut mit der Option *#FIXED*	Attribut einer Relation, konstanter Attributwert wird übernommen

Tabelle 2.4.: Abbildung der DDT auf die Datenbankstruktur, vgl. [35] geändert

2.7.4. Mapping des XML-Schemas

Mögliche Abbildungsregeln für das Exportieren von XML-Schemata in eine relationale Datenbank werden in der folgenden Auflistung zusammengefasst:

- Komplexe Elemente werden zu Relationen
- Einfache Elemente werden zu Attributen mit skalaren DB-Typen
- Sequenzen von Elementen werden zu Tupel
- XML-Schema-Komponenten, die optional sind können Nullwert annehmen
- Attribute der substitutionGroup, extension oder restriction
  - entweder in die Kind- oder Elterntabelle integrieren
  - oder in einer neuen Tabelle mit 1:1-Beziehung zur erbenden Tabelle speichern

Das O/R-Mapping des XML-Schema sieht vor die Objektstrukturen des XML-Schemas auf die relationale Datenbank abzubilden (s. Abb. 2.20). Dazu werden aus dem XML-Schema die komplexen Typen in Klassen bzw. Interfaces umgewandelt und eine Mapping-Datei generiert, welche Regeln für die Abbildung enthält. Aus der Mapping-Datei können dann über die DDL (s. Abschn. 2.2.2) die Tabellenstrukturen (Datenbankschema) erzeugt werden (s. Abb. 2.20). Bei dem Import von Daten (XML-Dokumente), die dem Datenbankschema entsprechen, müssen die Klassen bzw. Interfaces geladen werden und die angeforderten Instanzen (XML-Elemente) dieser Klassen durch Anwendung der Mapping-Datei erstellt werden (unmarshal). Im Anschluss werden aus diesen Objekten DML-Statements (vgl. 2.2.2) generiert.

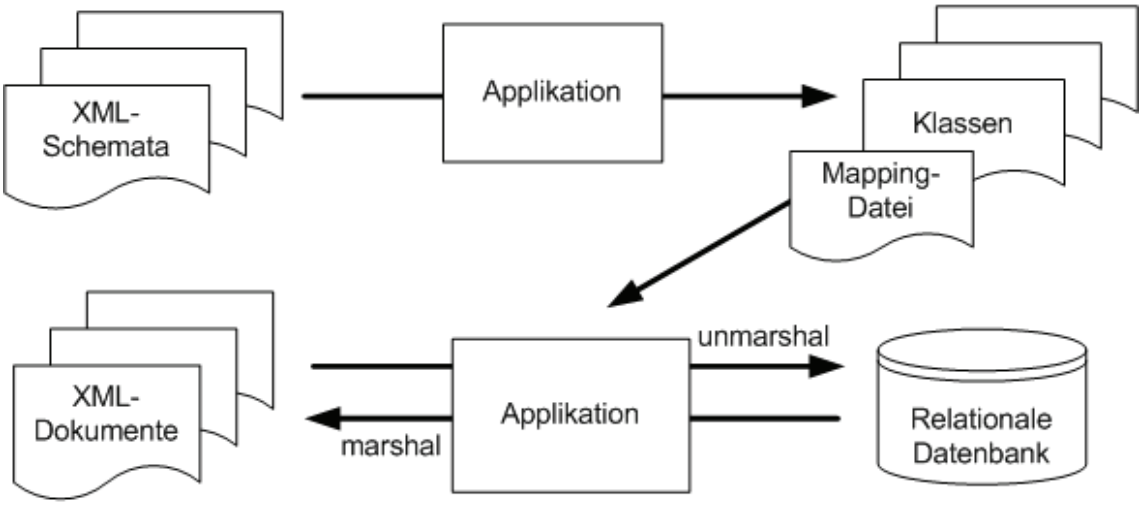


Abbildung 2.20.: O/R-Mapping-Prinzip

Das Listing 2.14 stellt ein einfaches Beispiel des objektrelationalen Mappings eines XML-Schemas nach der kanonischen Abbildung dar.

```
1  <?xml version="1.0" encoding="UTF-8"
       standalone="no"?>
2  <xs:schema xmlns:xs="http://www.w3.org
       /2001/XMLSchema"
3   xmlns="http://www.paper-store.de/angebot
       "
4   targetNamespace="http://www.paper-store.
       de/angebot"
5   elementFormDefault="qualified">
6    <xs:element name="Angebot">
7      <xs:complexType>
8        <xs:sequence>
9          <xs:element ref="Preis"/>
10         <xs:element ref="Artikel"
               minOccurs="1" maxOccurs="
               unbounded"/>
11       </xs:sequence>
12       <xs:attribute name="id" type="
             xs:int" use="required"/>
13     </xs:complexType>
14   </xs:element>
15   <xs:element name="Artikel">
16     <xs:complexType>
17       <xs:sequence>
18         <xs:element name="Name" type="
               xs:string"/>
19         <xs:element name="Hersteller"
               type="xs:string"/>
20         <xs:element ref="Preis"/>
21       </xs:sequence>
22       <xs:attribute name="id" type="
             xs:int" use="required"/>
23     </xs:complexType>
24   </xs:element>
25   <xs:element name="Preis" type="
        xs:double"/>
26 </xs:schema>
```

Listing 2.14: XML-Schema

Klassen	**Tabellen**
Klasse Angebot {	Tabelle Angebot {
int id;	Attribut id INTEGER (PK)
int preis;	Attribut preis INTEGER
Artikel[ ] a;	}
}	
Klasse Artikel {	Tabelle Artikel {
int id;	Attribut id INTEGER (PK)
String name;	Attribut name VARCHAR
String hersteller;	Attribut hersteller VARCHAR
int preis;	Attribut preis REAL
}	Attribut angebot_id INTEGER (FK)
	}

3. Analyse

In diesem Kapitel werden die Problemstellung analysiert und die Anforderungen herausgestellt. Weiterhin wird das Konzept des objektrelationalen Mappings im Hinblick auf die Vorgehensweise der Abbildung von DTD und XML-Schema auf die relationale Datenbank anhand von Beispielen untersucht. Danach werden einige vorhandene freie und proprietäre Programme und Technologien analysiert und geprüft. Dabei geraten OpenSource-Tools in den Vordergrund, weil diese laut Aufgabenstellung anzustreben sind.

Für das Projekt stellen sich laut Problemstellung drei Teilprobleme heraus, wobei zwei Teilprobleme dem Gebiet O/R-Mapping zugeordnet werden können.

- Objektrelationales Mapping
  - XML-Schema (xsd) in ein relationales Datenbankschema exportieren
  - XML-Dokumente in das angelegte relationale Datenbankschema importieren
- Layerdefinitionen im WebGIS *kvwmap* generieren

3.1. Anforderungen

Die Anforderungen für das Objektrelationale Mapping werden in den Tabellen 3.1.1 und 3.1.1 auf der nächsten Seite herausgestellt. So wird eine bidirektionale Transformation nicht notwendiger Weise vorausgesetzt. Außerdem wird bei dem Generieren des Datenbankschemas die Bedingung getroffen, dass dieses bei einer erneuten Generierung nicht aktualisiert bzw. versioniert, sondern ersetzt wird. Die Implementierung des dritten Teilproblems soll eine Erweiterung für das WebGIS-Framework *kvwmap* sein und die Darstellung der Layer aus dem Datenbankschema ermöglichen. Somit muss dieses unabhängig von den beiden anderen Programmen betrachtet werden. Die Layer in der Mapdatei sollen aus dem generierten Datenbankschema erzeugt werden. Dabei kann die benötigte Funktion in dem vorhandenen generischen Layereditor integriert werden. Die Applikation *kvwmap* besitzt eine Graphische Oberfläche, wobei diese auch verwendet werden sollte.

3.1.1. O/R-Mapping-Tools

Datenbank-Typ	Objektrelational (PostgreSQL/PostGIS)
Transformation	unidirektional, d.h. Transformation von XML-Schema nach DB oder XML-Dokument nach DB
Lizenz	Vorzugsweise Open-Source
Einschränkung zum Datenbankschema-Generator	Keine Unterstützung von Versionierung, somit Ersetzung des alten DB-Schemas

Tabelle 3.1.: Funktionale Anforderungen

Bedienbarkeit	Graphische Oberfläche ist nicht unbedingt notwendig, da nur wenige Parameter wesentlich sind, weitere Einstellungen können in einer Konfigurationsdatei vorgenommen werden, Kommandozeilen-Programme lassen sich für Administratoren leichter handhaben, bspw. Eingliederung in Batch-Prozesse
Portierbarkeit	Programme sollen ohne aufwendige Installation auf jeder Plattform verwendet werden können

Tabelle 3.2.: Nichtfunktionale Anforderungen

3.1.2. Layerdefinitionsgenerator

Anwendung	Generierung und Darstellung der Layer aus den Datenbankrelationen, evt. Benutzung des vorhandenen generischen Layereditors
Bedienung	Graphische Web-Oberfläche des *kvwmap*

Tabelle 3.3.: Anforderungen

3.2. Stand der Technik in GIS-Applikationen

Geodatenbanken sind objektrelational oder objektorientiert und der Wunsch nach Speicherung von GML liegt nahe. Die Analyse von GIS-Applikationen hat aber gezeigt, dass die Entwicklung von freien ORM-Tools im Bereich der Geoinformatik noch keinen generischen Ansatz hervorgebracht hat, der jedes GML-Anwendungsschema „on the fly“ auf eine relationale Datenbank abbilden kann. Die Komplexität von GML ist nicht zu verachten und die Anwendungsschemata werden meistens so angepasst, das die Interoperabilität erschwert wird (vgl. [81]). Die Firma ESRI hat sich mit ArcGIS-Lösungen auf die Verwaltung von INSPIRE-Daten spezialisiert (vgl. [3]) und das Unternehmen Galdos Systems Inc. bietet mit dem INtune SDI Framework (vgl. [19]) eine Lösung an. Bei den freien GIS-Tools ist, abgesehen von deegree und GDAL aber wenig passiert in Bezug auf ORM-Tools für GML.

3.2.1. kvwmap

Das Open-Source WebGIS-Framework *kvwmap* war ursprünglich für Katasterinformationen vorgesehen. Im Laufe der Zeit sind aber viele Fachschalen-Erweiterungen vorgenommen worden, sodass es insbesondere für Kommunen eine sehr gute Alternative zu herkömmlichen kommerziellen Systemen (z.B. ArcGIS und Smallworld) darstellt.

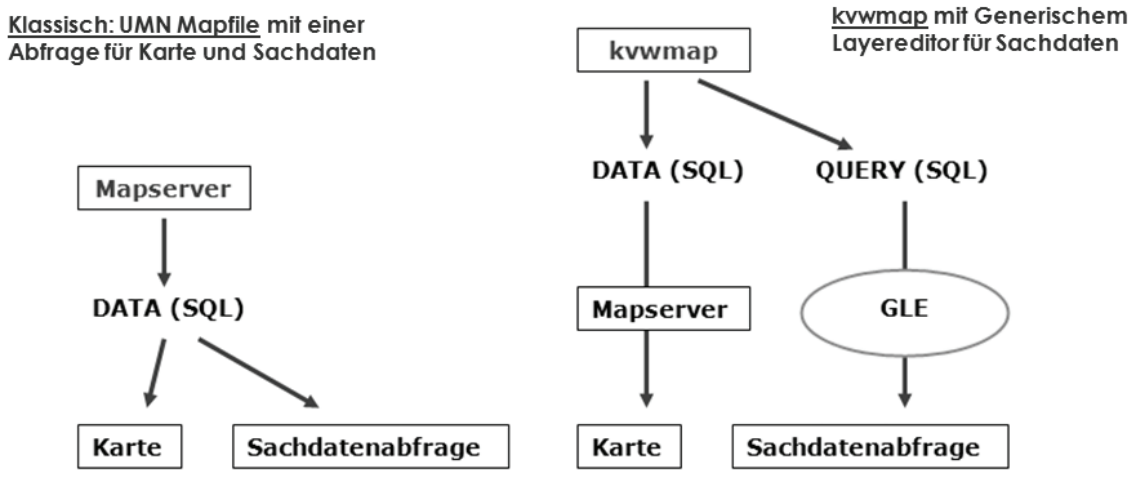


Abbildung 3.1.: Mapdatei-Generierung: Konventionell und GLE im kvwmap [37]

Ein Modul des *kvwmap* ist der Generische Layereditor (GLE). Dieser hat im Gegensatz zum konventionellen statischen OWS[1] den Vorteil, dass die Layer in dem Programm vom Benutzer erstellt, verändert und abgefragt werden können (s. Abb. 3.1), das bestrifft auch ihre Geometrie. Insbesondere die Sachdatenabfrage ist dadurch sehr flexibel. Die Daten, die von kvwmap-Anwendungen erfasst werden, bestehen öfter aus GML-Dokumenten. Ein O/R-Tool wird nicht unbedingt verwendet, die Daten werden noch manuell abgebildet.

[1] OWS: OGC Web Services (z.B. WMS und WFS)

3.2.2. deegree

Deegree ist ein quelloffenes WebGIS-Framework, welches insbesondere von der Firma lat/lon angeleitet wird. Es beherbergt zahlreiche GIS-Funktionalitäten, darunter auch die Verarbeitung von GML.

xplan-manager

Der *xplan-manager* ist eine Java-Anwendung, welche das Format XplanGML in den Versionen 2.0 und 3.0 verarbeiten kann. Es entstammt aus dem *xplan-node*, einer Geodateninfrastruktur für XPlanungs-Daten (vgl. [82]). Das Programm wird über ein Skript in der Kommandozeile gestartet. Die Datenbank-Konfiguration (JDBC) wird in der Datei *jdbc_connections.xml* vorgenommen. Zu beachten sind einige Randbedingungen (vgl. [81]). Das XplanGML-Dokument muss in dem ZIP-Format mit sämtlichen Plänen, Anhängen o.ä. vorliegen und xplan.gml heißen.

```
Benutzung: xplan-manager -list | -delete <planid> | -import [-force]
<xplanarchiv> [CRS] | -export <planid> [<verzeichni>]

Beispielaufruf: xplan-manager -import Landschaftsplan_3_0.zip
```

Ein wesentliches Merkmal des Programms ist die Überprüfung der Geometrie (s. Abb. 3.2).

```
>xplan-manager -import Landschaftsplan_3_0.zip
- Analyse des XPlan-Archivs ('Landschaftsplan_3_0.zip')...OK.
- Analyse des Dokuments...OK [328 ms]: XPLAN_3, LP_Plan, EPSG:31466
- Schema-Validierung...OK [889 ms]
- Einlesen der Features (+ Geometrievalidierung)...Fehler.

Geometrie-Fehler: 1
 - LinearRing (Ende in Zeile 30047, Spalte 30): Selbst³berschneidung.

1 Geometrie-Fehler, 0 Geometrie-Warnung(en). Hinweis: Sie k÷nnen das Importieren des Plans mit der
 --force erzwingen.
```

Abbildung 3.2.: xplan-manager: Geometrie-Fehler

deegree-webservices

Die Projektgruppe von *deegree*, welches insbesondere von der Firma lat/lon angeleitet wird, entwickelt einen generischen Ansatz für das O/R-Mapping von GML und stellt diesen in der Beta-Version *deegree-webservices-3.2* (vgl. [13]) bereit. Dabei handelt es sich um einen Extract-Transform-Load-Ansatz, bei dem deegree3 alle Prozesse übernimmt. Ziel ist es, die Konzepte des *xplan-nodes*, *inspire-nodes* und anderen Geodatenlösungen von *deegree* zu vereinen um ein generisches ORM-Tool zu erhalten. Die Web-Applikation *deegree-webservices* wird mit einem Apache Tomcat Webserver geliefert und über ein Skript (start-deegree.bat (Windows) oder start-deegree.sh (Linux)) gestartet.

Zu beachten sind bei dem O/R-Mapping eines GML-Anwendungsschemas oder Teile aus diesem folgende Schritte:

1. Laden eines eigenen oder offiziellen Workspace
2. Konfigurieren der Datenbankverbindung (JDBC)
3. Einrichten des *deegree-Feature-Store*
4. Erzeugen der Datenbanktabellen

Nach dem Start des Web-Servers wird ein Browser geöffnet (empfohlen wird Mozilla Firefox) und die Anwendung entweder lokal (http://localhost:8080) oder nach Tomcat-Anpassung über eine Domäne angewählt. Der Benutzer muss sich einloggen (default: Passwort = 'deegree'). Anschließend stehen dem Benutzer eine Reihe von Funktionen zur Verfügung, die sich in dem Menü auf der linken Seite der graphischen Oberfläche befinden (s. Abb. 3.3).

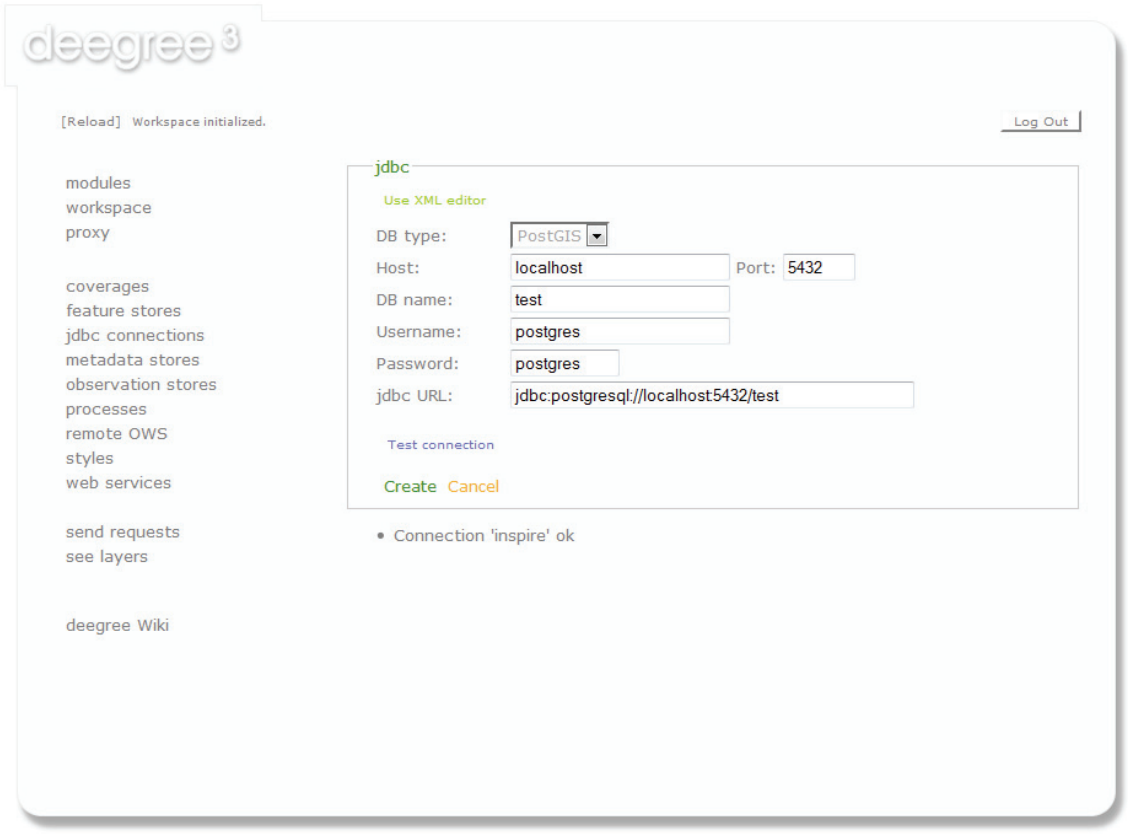


Abbildung 3.3.: deegree-webservices: GUI

Über den *Loader* (s. Abb. 3.4) im *feature store* können GML-Dokumente in die Datenbank eingelesen werden. oder über einen WFS-T können Daten eingelesen, verändert und gelöscht werden (s. Abb. 3.5).

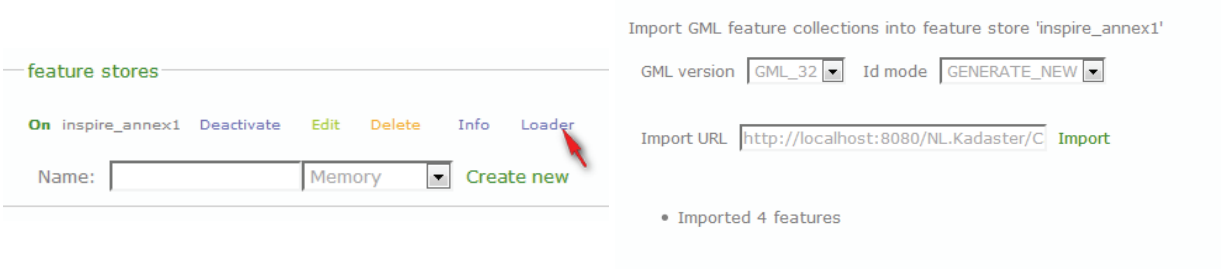


Abbildung 3.4.: Import eines niederländischen INSPIRE konformen Kataster-Demodatensatzes über den *Loader*

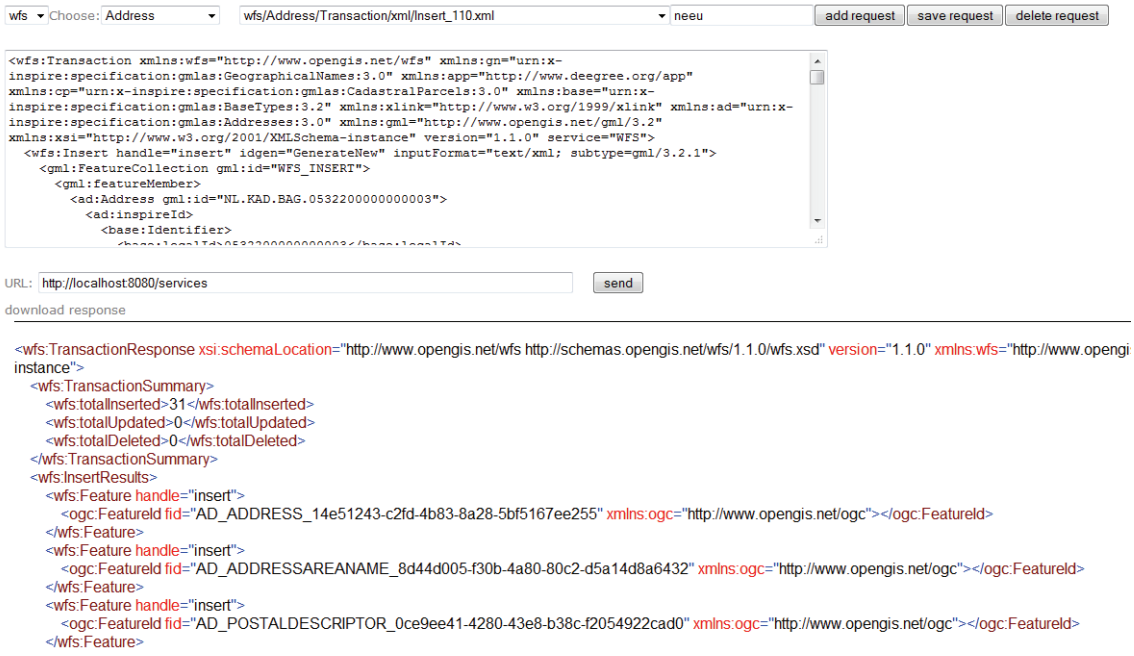


Abbildung 3.5.: Import von INSPIRE konformen Adressen über den WFS-T

3.2.3. PostNAS

PostNAS (vgl. [54]) ist eine Entwicklung der WhereGroup und Modul der Bibliothek für Raster- und Vektorverarbeitung GDAL. „PostNAS ist eine Schnittstelle, mit der als NAS-Datei gelieferte ALKIS und ATKIS-Daten in OGC-konforme Geodateninfrastrukturen überführt werden können, also ein "NAS2WKT"-Konverter“ (vgl. [55]). Das Modul wird über die Kommandozeile mittels des GDAL-Programms ogr2ogr aufgerufen.

Beispiel:

```
ogr2ogr -f "PostgreSQL" "PG:host=localhost user=postgres port=5432 dbname=alkis_test" -a_srs EPSG:25833 BeispielNAS.gml
```

PostNAS kann mit jeder Art von GML-Dokument umgehen, aber nicht immer alle Attribute des Schemas werden abgebildet. Das Programm generiert aus einem GML-Dokument eine gfs-Datei, die die Features des Dokuments enthält. Aus der gfs-Datei wird dann die DDL erzeugt und ausgeführt. Im Datenbankschema werden aber keine relationalen Beziehungen hergestellt, d.h. keine Fremdschlüssel gesetzt. PostNAS generiert lediglich Primärschlüssel. In der Relation *alkis_beziehungen* werden aber trotzdem die Beziehungen aus dem GML-Dokument abgebildet (s. Abb. 3.6). GML-Dokumente, die über PostNAS eingelesen werden sollten vorher validiert werden, denn PostNAS besitzt keinen Validierungs-Mechanismus. Der XML-Parser bricht bspw. nur ab, wenn nicht korrespondierende Tags gefunden werden.

	ogc_fid [PK] serial	beziehung_von character varying	beziehungsar character var	beziehung_zu character varying
1	1	DERP123400000001	istTeilVon	DERP123400000hAH
2	2	DERP123400000002	istTeilVon	DERP123400000hAI
3	3	DERP123400000003	istTeilVon	DERP123400000hAJ
4	4	DERP123400000004	istTeilVon	DERP123400000hAK
5	5	DERP123400000005	istTeilVon	DERP123400000hAL
6	6	DERP123400000006	istTeilVon	DERP123400000hAM

Abbildung 3.6.: Auszug aus der Relation *alkis_beziehungen*

Für das Betriebssystem Windows werden immer neue *Binaries* zum freien Download angeboten (vgl. [54]). Die Kompilierung in Linux mit minimalen Treibern kann folgender Maßen Erfolgen:

```
 ./configure -prefix=/usr/local  -with-threads  -with-ogr  -with-geos
-without-libtool  -with-libz=internal  -with-libtiff=internal
-with-geotiff=internal  -without-gif  -with-pg  -without-grass
-without-libgrass  -without-cfitsio  -without-pcraster -without-netcdf
-without-png  -without-jpeg  -without-gif  -without-ogdi  -without-fme
-without-hdf4  -without-hdf5  -without-jasper  -without-ecw
-without-kakadu  -without-mrsid  -without-jp2mrsid  -without-bsb
-without-grib  -without-mysql  -without-ingres  -with-xerces
-without-expat  -without-odbc  -without-curl  -without-sqlite3
-without-dwgdirect  -without-panorama  -without-idb  -without-sde
-without-perl  -without-php  -without-ruby  -without-python
-without-ogpython  -with-hide-internal-symbols
```

Zusammenfassung

PostNAS bietet einen einfachen Mechanismus um GML auf die relationale Datenbank abzubilden. Es wird aber nur die GML-Instanz abgebildet. Die Abbildung eines komplexen Schemas ist somit nicht möglich.

3.3. Freie O/R-Mapping-Tools

In diesem Abschnitt werden vorhandene nicht-kommerzielle Produkte untersucht und deren mögliche Verwendung geprüft. Dabei muss beachtet werden, dass es sich bei den Produkten nicht immer um eigenständige Programme handelt. Oftmals wird eine anwendungsorientierte Middleware angeboten. Der englische Begriff wird meistens im Zusammenhang mit verteilten Systemen verstanden und bedeutet zu deutsch eine Zwischenanwendung. Für den Terminus existiert keine standardisierte Definition:

„Als Middleware wird eine Software-Schicht zwischen (Netzwerk-) Betriebssystem und Applikationsebene bezeichnet“ [32].

Somit muss ein Programmieraufwand betrieben werden um die Middleware zu benutzen. Ein klassisches Beispiel für eine Datenbank-Middleware stellt z.B. der ODBC-Treiber oder das Pendant dazu, der JDBC-Treiber, dar (vgl. [20]).

3.3.1. XML-DBMS

XML-DBMS ist eine Open-Source Middleware, die von dem Amerikaner Ronald Bourret entwickelt wurde. Diese Software wird in den Programmiersprachen Java und Perl angeboten. Sie unterstützt das objektrelationale Mapping und kann Daten zwischen XML-Dokumenten und relationalen Datenbanken transferieren (vgl. [6]). Das Programm ist bidirektional, d.h. der Transfer kann vom Dateisystem zur Datenbank und von der Datenbank zum Dateisystem erfolgen.

Arbeitsweisen

- DTD-Datei in Map-Datei exportieren
- DTD-Datei in Relationales Datenbankschema exportieren
- Relationales Datenbankschema in Map-Datei exportieren
- Relationales Datenbankschema in DTD-Datei exportieren

Probleme Version 1

- PostgreSQL: Die Reihenfolge der generierten Relationen ist richtig (Angebot, Artikel), aber das Semikolon fehlt nach jedem CREATE-Statement
- Ansatzweise Unterstützung für W3C-Schema, wird aber nicht mehr gepflegt

Probleme Version 2 alpha 3

- Keine Unterstützung für W3C-XML-Schema
- PostgreSQL: Tabellenstrukturen werden nicht in korrekter Abfolge gespeichert (Reihenfolge der Relationen, bspw. Artikel, Angebot anstatt Angebot, Artikel)
- MySQL: Bei der Tabellendefinition fehlt die Anzahl der Zeichen bei dem Datentyp VARCHAR, z.B. VARCHAR(255), muss ab mySQL-Version 5.5.9 gesetzt sein
- PostgreSQL: Benutzung manage.bat: Konfiguration "DataSourceClass" muss in der Datei db.props auskommentiert werden
- PostgreSQL: Benutzung transfer.bat: Konfiguration "DataSourceClass" wieder einkommentieren
- weitere Probleme siehe [7], Abschnitte 3.7 und 3.8

Zusammenfassung

XML-DBMS ist plattformabhängig, in Modulen aufgebaut und nach einiger Einarbeitung leicht zu verwenden. Die Version 2.0 (Alpha 3) hat im Zusammenhang mit dem JDBC-Postgres-Treiber für einige Unstimmigkeiten gesorgt, d.h. händische Eingriffe sind notwendig. Im Vergleich zur Version 1 ist die Benutzung für den Anwender erheblich erleichtert worden durch Batch-Files, die kompilierte Klassen (MapManager / Transfer) aufrufen. XML-DBMS unterstützt nur teilweise XML-Schema, daher kann es in diesem Projekt nur durch Erweiterung des Programms benutzt werden.

3.3.2. XSLT

XSLT in Verbindung mit XPath kann für die Erzeugung der DDL aus einem XML-Schema verwendet werden. Dabei können einfache XML-Schemata schnell verarbeitet werden. Die Verarbeitung von komplexen Schemata, bspw. GML-Anwendungsschemata erweist sich aber als sehr problematisch, weil z.B. jede Art von Referenz und substitutionGroup aufgelöst werden muss. Das XSLT-Beispiel (s. Anhang B.1) zeigt die Verarbeitung eines einfachen XML-Schemas und die Generierung der DDL in PostgreSQL (s. Anhang B.2).

3.3.3. XSOM

XSOM (XML Schema Object Model) ist eine freie Java-Bibliothek zum Parsen von XML-Schema-Dateien (vgl. [83]). Zur Installation muss der CLASSPATH der Java-Runtime Umgebung um diese Bibliothek (jar-Datei) erweitert werden. Bei Verwendung einer Entwicklungsumgebung wie z.B. NetBeans (vgl. [46]) oder Eclipse (vgl. [15]) geschieht dieser Vorgang automatisch nach Einbindung der Bibliothek. Die Klassen, die für den Zugriff auf das XSOM bereitstehen, sind fast ausschließlich abstrakt bzw. als *Interface* deklariert (s.

Anhang C). Das hat den Vorteil, dass eine Mehrfachvererbung realisiert werden kann (vgl. [39]). Die Unterscheidung der XML-Schema-Komponenten *complexType* und *simpleType* lässt sich somit über das Interface-Konzept realisieren. Die Methodendeklarationen der Klasse *XSType* werden an ihre Subklassen *XSSimpleType* und *XSComplexType* vererbt. XSOM wurde streng nach den Vorgaben des W3C implementiert. Das wird deutlich bei dem Vergleich des *XSOM-Interface-Modell* (s. Anhang C) mit dem *XML Schema Component Data Model* des W3C (s. Anhang D).

3.3.4. XML To DDL

XML To DDL ist ein Projekt des Open-Source-Vermittlers *berlios* und stellt eine Menge von Python-Programmen bereit um ein XML-basiertes Schema in eine Datenbank zu laden und um von einer Datenbank ein XML-basiertes Schema zu erhalten (vgl. [80]). Dabei muss hier ausdrücklich darauf hingewiesen werden, dass es sich dabei nicht um ein W3C-XML-Schema handelt, und dass somit auch keine Validierung des Schemas stattfinden kann. Viel eher ist diese XML-Repräsentation als Mapping-Definition zu verstehen, ähnlich wie sie auch in dem Projekt *Hibernate* (s. Abschn. 3.3.6) als hbm-Format gegeben ist.

```
1  <schema>
2      <table name="students" fullname="List of Students"
3          desc="List of students with their full names">
4          <columns>
5              <column name="id" fullname="Primary Key" type="integer" key="1"
6                  desc="Primary key for the table"/>
7              <column name="student_name" fullname="Student Name" type="varchar" size
                   ="80"
8                  desc="The full name of the student"/>
9          </columns>
10     </table>
11 </schema>
```

Listing 3.1: XML-basiertes Schema [80]

Das Kommandozeilen-Programm *xml2ddl* wird dazu benutzt um aus einem XML-basierten Schema (s. Listing 3.1) DDL zu erzeugen, eine Python-Umgebung muss installiert sein.

Aufruf über die Kommandozeile:

```
Beispielaufruf: python xml2ddl.py -b postgres ..\www\schema1.xml
```

Ausgabe der DDL für PostgreSQL:

```
DROP TABLE students;
CREATE TABLE students (
       id integer, student_name varchar(80),
```

```
       CONSTRAINT pk_students PRIMARY KEY (id)
);
COMMENT ON TABLE students IS
       'List of students with their full names';
COMMENT ON COLUMN students.id IS 'Primary key for the table';
COMMENT ON COLUMN students.student_name IS 'The full name of the
student';
```

Funktionen

- Datenbank-Unterstützung für PostgreSQL, MySQL, Oracle, und Firebird
- Anwendung von *dictionaries* um XML-Klassen zu bilden, die mehrfach verwendet werden können, bspw. Definition des Primärschlüssels
- *xml2ddl* : XML-basiertes Schema in Datenbankschema umwandeln
- *diffxml2ddl* : Vorhandenes Datenbankschema aktualisieren
- *downloadXml* : Datenbankschema als XML-basiertes Schema exportieren
- *xml2html* : XML-basiertes Schema in einem dokumentierten HTML-Format speichern

Zusammenfassung

XML2DDL ist ein zuverlässiges Tool um vordefinierte XML-basierte Schemata in Datenbankschemata umzuwandeln. Für die geforderte Verwendung müsste aber zunächst das zu bearbeitende XML-Schema in dieses Mapping-Format konvertiert werden. Anschließend könnte das Tool für das objektrelationale Mapping genutzt werden. Im weiteren Schritt ergeben sich aber Probleme bei dem Einlesen von XML-Dokumenten (Daten) in das zuvor erstellte Datenbankschema, denn hierfür ist ein Programmieraufwand von Nöten. Das Programm wird auch laut Entwickler eher zum Testen benutzt.

3.3.5. XSD2DB

XSD2DB ist nur für Microsoft SQL Server und MSDE OleDB kompatible Datenbanken zu verwenden (vgl. [84]). Daher wird es hier nicht weiter betrachtet.

3.3.6. Hibernate / Hibernate-Spatial

Hibernate ist ein in Java geschriebenes ORM-Framework, dass die Speicherung von Java-Objekten in relationalen Datenbanken ermöglicht. Durch die Erweiterung Hibernate-Spatial können das auch Geometrieobjekte (JTS) sein. Das O/R-Mapping von XML-Dateien ist noch in der Anfangsphase (vgl. [24]). Nach Erstellung einer Mapping-Datei (*.hbm.xml, s. Ann. 3.2) kann das Hibernate-Tool hbm2ddl dafür genutzt werden um aus der Mapping-Datei DDL zu erzeugen.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
5  <hibernate-mapping>
6
7  <class entity-name="Lehrer" schema="schulklasse"
8       table="LEHRER" node="lehrer">
9    <id name="id" column="lehrerID" node="id" type="long">
10         <generator class="sequence">
11             <param name="sequence">lehrer_seq</param>
12         </generator>
13   </id>
14
15  <property name="nname" column="NNAME" node="nname" type="string"/>
16  <property name="vorname" column="VORNAME" node="vorname" type="string"/>
17 </class>
18
19 <class entity-name="Schueler" schema="schulklasse"
20      table="SCHUELER" node="schueler">
21   <id name="id" column="schuelerID" node="id" type="long">
22         <generator class="sequence">
23             <param name="sequence">schueler_seq</param>
24         </generator>
25   </id>
26
27   <many-to-one name="lehrerId" column="LEHRER_ID" node="lehrer/@id"
28   embed-xml="false" entity-name="Lehrer"/>
29   <property name="nname" column="NNAME" node="nname" type="string"/>
30   <property name="vorname" column="VORNAME" node="vorname" type="string"/>
31 </class>
32 </hibernate-mapping>
```

Listing 3.2: Beispiel: Hibernate-Mapping-Datei schulklasse.hbm.xml

3.3.7. JAXB

Das Project Java API for XML Processing (JAXP) ist eine Entwicklung der Firma Sun Microsystems (jetzt Oracle) und bietet Schnittstellen um in Java XML zu verarbeiten. JAXB enthält DOM- und SAX-Parser. Außerdem können mit JAXB über den Binding Compiler (xjc) Klassen aus XML-Schemata und entsprechende Java-Objekte aus XML-Dokumenten erzeugt werden (s. Abb. 3.7).

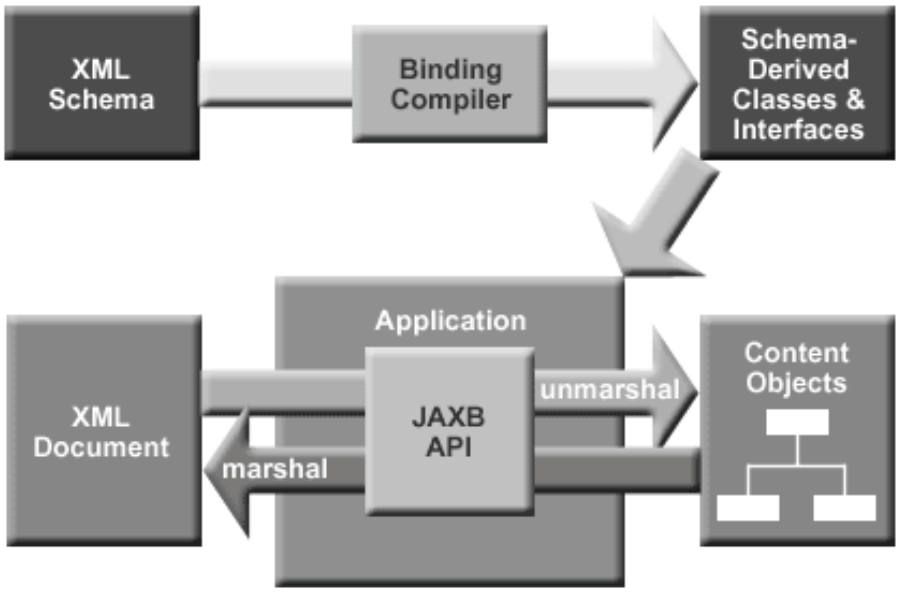


Abbildung 3.7.: JAXB-Modell [33]

3.3.8. HyperJAXB

HyperJAXB ist eine Entwicklung des Softwareingenieurs A. Valikov und erweitert JAXB in der Hinsicht, dass JAXB-Objekte in relationalen Datenbanken persistiert werden können (vgl. [27]). Hierfür ist keine Mapping-Datei erforderlich. Für ein direktes O/R-Mapping werden Ant oder Maven-Projekte angeboten (vgl. [27]). Zur Generierung von DDL werden lediglich XML-Schema-Dateien benötigt. Ein XML-Schema wird mittels JAXB bzw. des JAXB Binding Compilers (xjc) in Java-Klassen exportiert. Diese können bspw. über das Hibernate-Tool *hbm2ddl* direkt in einer PostgreSQL-Datenbank als Relationen gespeichert werden. Während der Laufzeit können anschließend korrespondierende XML-Dokumente über den „unmarshal“-Vorgang in JAXB-Objekte und in die Datenbank eingelesen werden. Die Beziehungen 1:1 und 1:N werden in der Datenbank über Fremdschlüssel abgebildet.

3.4. Proprietäres O/R-Mapping-Tool von Altova

Die Anwendung XMLSpy (Version 2011 rel.2) der Firma Altova bietet ein ORM-Tool für XML. Nach Öffnen eines XML-Schemas und Betätigung des Buttons Schema kann über den Reiter „Convert“ und den Eintrag „Convert DB Structure from XML Schema“ DDL aus dem Schema erzeugt werden (s. Abb. 3.8). Die Datenbankschnittstellen sind ADO und ODBC. GML kann in Bezug auf das O/R-Mapping nicht verarbeitet werden.

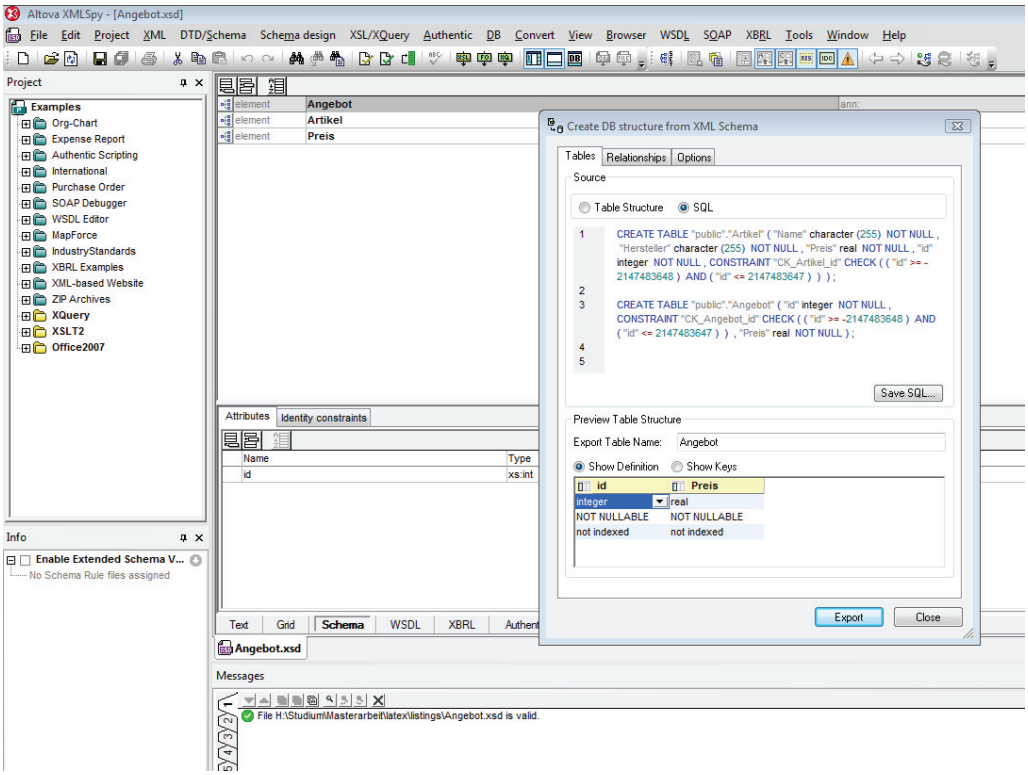


Abbildung 3.8.: Generierung der DDL aus einem XML-Schema

MapForce ist eine Erweiterung für Altova XMLSpy. Mittels MapForce kann ein XML-Dokument auf ein vorhandenes Datenbankschema abgebildet werden. Es bietet eine graphische Oberfläche und der Anwender kann Datenbanktabellen sowie XML-Dokumente laden. Durch Verbinden der Attribute der Tabellen untereinander und der Attibute des XML-Dokument per „drag and drop“, werden die Beziehungen gesetzt (s. Abb. 3.9). MapForce generiert nach Aufruf des Buttons „Output“ ein SQL-Script, welches der Benutzer ausführen kann (s. Listing 3.3). Die Anwendung kann u.a. Java, C, C# usw. Code generieren um das für die Zukunft zu automatisieren. MapForce ist bidirektional und kann auch von einer Datenbank ausgehend XML exportieren.

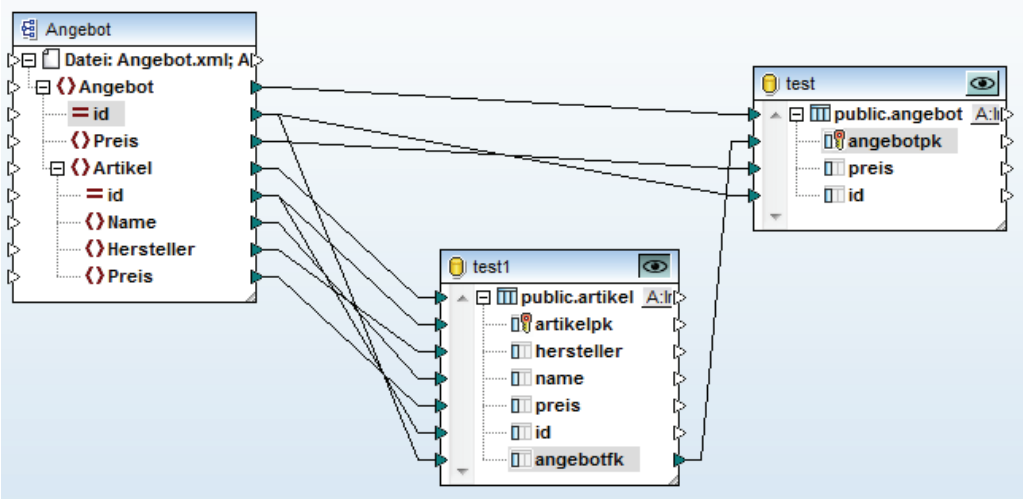


Abbildung 3.9.: Mapping Angebot

```
/*
Diese SQL-Anweisungen sind nur für die Vorschau bestimmt und können in einem anderen SQL-Abfrage-Tool nicht ausgeführt werden!
Zur Ausführung dieser Anweisungen verwenden Sie die Funktion "SQL-Script ausführen" im Menü "Ausgabe".
Stellen Sie die Verbindung zur Datenbank mit Hilfe des folgenden Connection String her:
DSN=PostgreSQL35W;UID=postgres;Pwd=postgres;DATABASE=test;Port=5432;BI=0;
*/

INSERT INTO "public"."angebot" ("angebotpk", "preis", "id") VALUES (100, '3.59', '100')

INSERT INTO "public"."artikel" ("artikelpk", "hersteller", "name", "preis", "id", "angebotfk") VALUES (1, 'Schumann', 'Bleistift', '1', '1', 100)

INSERT INTO "public"."artikel" ("artikelpk", "hersteller", "name", "preis", "id", "angebotfk") VALUES (2, 'mpaper', 'Collageblock', '2.59', '2', 100)
```

Listing 3.3: Angebot-SQL

4. Entwurf

4.1. Anwendungsfälle

Die Abb. 4.1 zeigt drei Anwendungsfälle. Zum einen handelt es sich um den DB-Schema-Generator und den DB-Importer. Zum anderen ist hier der Layerdefinitionsgenerator im WebGIS *kvwmap* gemeint. Das objektrelationale Mapping von XML-Schemata kann durch Ausführung eines Programms von einem Benutzer oder einem Autostart-Prozess angewendet werden. Dasselbe gilt für den DB-Importer, der XML-Dokumente in eine PostgreSQL-Datenbank einlesen soll. Der Layerdefinitionsgenerator soll im WebGIS-Framework *kvwmap* implementiert werden und dazu dienen aus relationalen Datenbanken Mapserver-Layer bzw. OWS-Layer zu erzeugen.

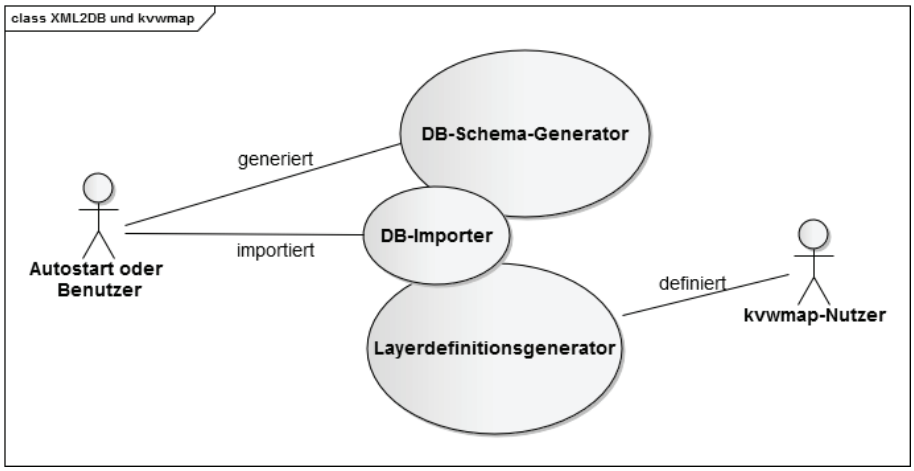


Abbildung 4.1.: Anwendungsfälle

4.2. Generischer Ansatz XML-Mapping

Jede Klasse wird auf eine Tabelle abgebildet, wobei der Tabellenname üblicher Weise dem Klassennamen entspricht. Um zu kennzeichnen, dass eine Klasse abstrakt ist, kann dem Bezeichner ein Unterstrich vorangestellt werden. Die Attribute werden in Tabellenattribute mit primitiven Datentypen abgebildet. Für die Verwandtschaftsbeziehungen kommt eine Meta-Tabelle namens *Klassen* zum Einsatz. Diese Tabelle enthält die ID der Klasse, den Klassennamen und die ID zu der Superklasse. Somit kann von einer Klasse die Superklasse und von einer Superklasse die Subklasse ermittelt werden. Die Beziehungen 1:1 und 1:N werden nach den Regeln des relationalen Modells aufgelöst.

Die Aufgabenstellung setzt sich aus drei Problemfeldern zusammen. Somit ergeben sich demnach drei Programme, ein DB-Schema-Generator, der XML-Schema in ein Datenbankschema konvertiert, ein DB-Importer, der XML-Dokumente in die Datenbank einliest und der Layerdefinitionsgenerator, der für die Darstellung der eingelesenen XML-Instanzen sorgt.

Der DB-Schema-Generator übernimmt die wichtigste Rolle in dem Geschäftsprozess, denn dieses Programm bestimmt das Verhalten der anderen Programme. Das Programm sollte zunächst das eingegebene XML-Schema validieren, d.h. auf Gültigkeit überprüfen (vgl. Abb. 4.2), denn wenn das XML-Schema nicht valide ist, kann der Parser nicht ordnungsgemäß arbeiten. Anschließend ist zu überprüfen, ob ein entsprechendes DB-Schema vorhanden ist. Trifft das nicht zu, kann das XML-Schema mittels Parser zergliedert werden, andernfalls wird das Programm beendet. Im weiteren Schritt müssen die XML-Schema-Komponenten verarbeitet und in Klassen umgesetzt werden, denn die Klassen werden für den Import der XML-Daten (s. Abb. 4.3) notwendigerweise benötigt. Währenddessen können die SQL-Statements für die Erzeugung des DB-Schemas erstellt werden (vgl. Abschn. 2.7.4). Schließlich wird das Datenbankschema erzeugt, bzw. die SQL-Statements ausgeführt.

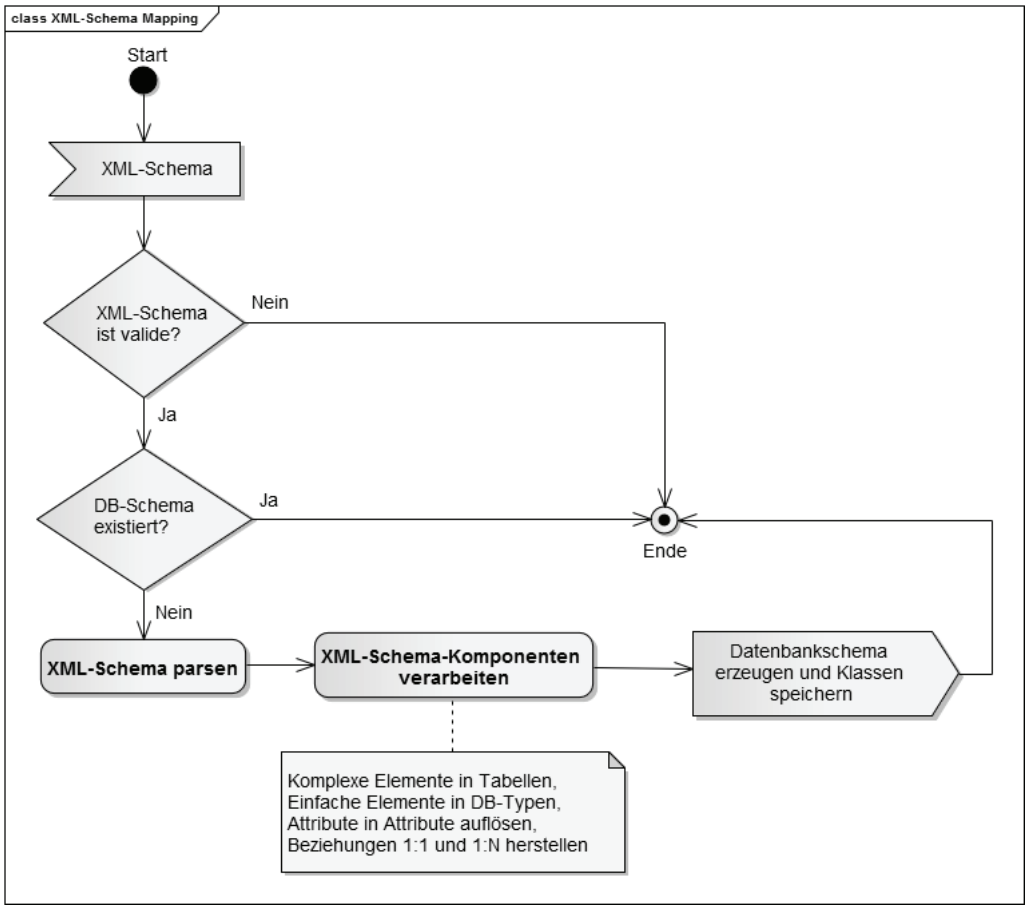


Abbildung 4.2.: DB-Schema-Generator

Eine Möglichkeit um Daten (XML-Dokument) in eine Datenbank zu importieren wird

im nachfolgenden Aktivitätsdiagramm (s. Abb. 4.3) aufgezeigt. Das XML-Dokument wird zunächst eingelesen und validiert, damit dieses auch zergliedert werden kann. Nach erfolgreicher Validierung sollte überprüft werden, ob ein entsprechendes Datenbankschema existiert. Wenn das nicht der Fall ist, wird das Programm abgebrochen. Im Anschluss daran sollte überprüft werden, ob das Dokument schon einmal in die Datenbank eingelesen worden ist. Wenn dieser Fall eintritt, wird das Programm abgebrochen, denn Redundanz soll vermieden werden. Im anderen Fall wird der Prozess fortgeführt und der Parser-Vorgang vorbereitet, indem die Klassen eingelesen werden. Nun wird das XML-Dokument zergliedert und aus den Daten Objekte erzeugt bzw. SQL-INSERT-Kommandos generiert. Danach wird das erstellte SQL-Skript auf die Datenbank angewandt.

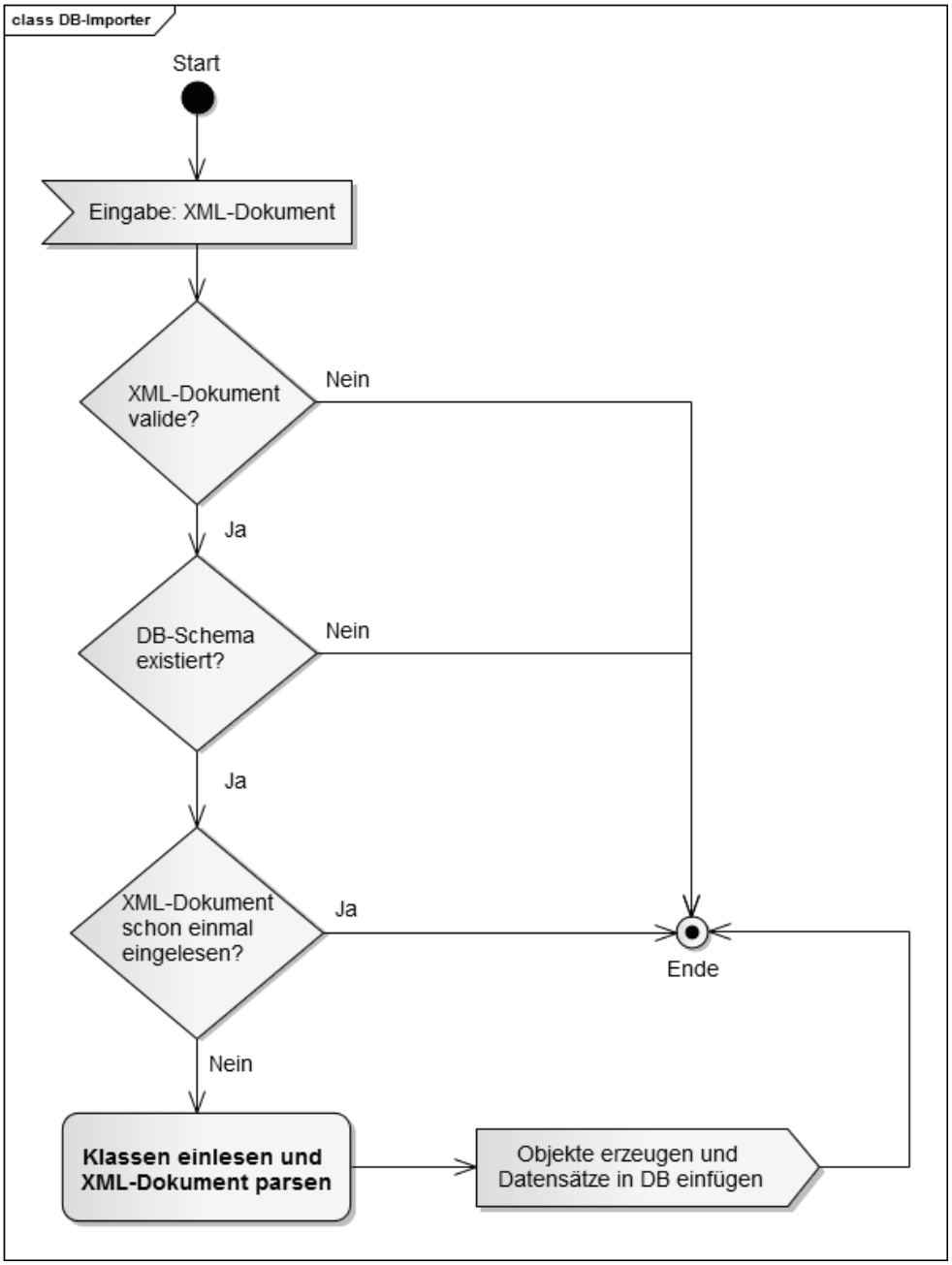


Abbildung 4.3.: DB-Importer

Der letzte Schritt im Ablauf des Geschäftsprozesses betrifft den *Layerdefinitionsgenerator*. Dieser soll im WebGIS *kvwmap* aus relationalen Datenbanktabellen, die eine Geometrie enthalten, OWS-Layer für den Mapserver erzeugen. In Abb. 4.4 ist eine mögliche Herangehensweise dargestellt. Zunächst wird die Datenbankverbindung zur PostGIS-Datenbank aufgebaut, im *kvwmap* ist das üblicherweise die Datenbank *kvwmapsp* mit eventueller Versionsnummer. Nach einem erfolgreichen Verbindungsaufbau sollte die Möglichkeit gegeben sein ein Datenbankschema auszuwählen (z.b. xplan_30). Die Anwendung muss nun alle darin enthaltenen Tabellen abfragen und in einer Iteration verarbeiten. Wenn es sich um eine Tabelle mit einer Geometrie handelt, wird diese in einer Datenstruktur gespeichert und für eine generische Mapdatei vorbereitet.

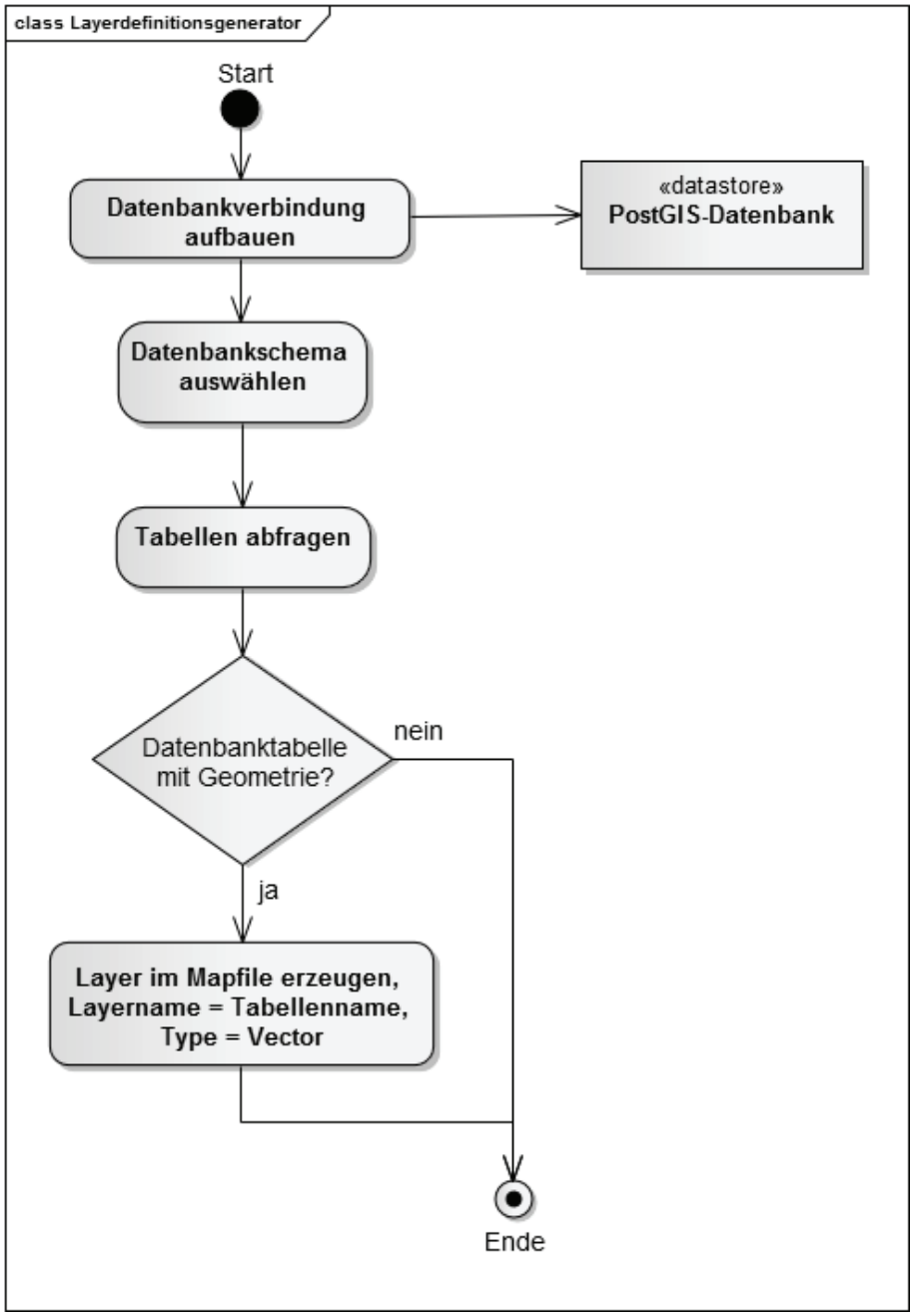


Abbildung 4.4.: Layerdefinitionsgenerator

5. Realisierung

5.1. Implementierung des Tools xsd2ddl

Nach Analyse der freien ORM-Tools wird eine Implementierung auf Basis von XSOM, Hibernate und Hibernate-Spatial vorgenommen.

5.1.1. Aufbau der Anwendung

Das entwickelte Java-Programm dient zum Generieren von Datenbankstrukturen aus XML-Schema-Dateien. Für das Parsen von XML-Schema-Dateien wird die Java-Bibliothek *XSOM* (XML Schema Object Model, s. Abschn. 3.3.3) verwendet.

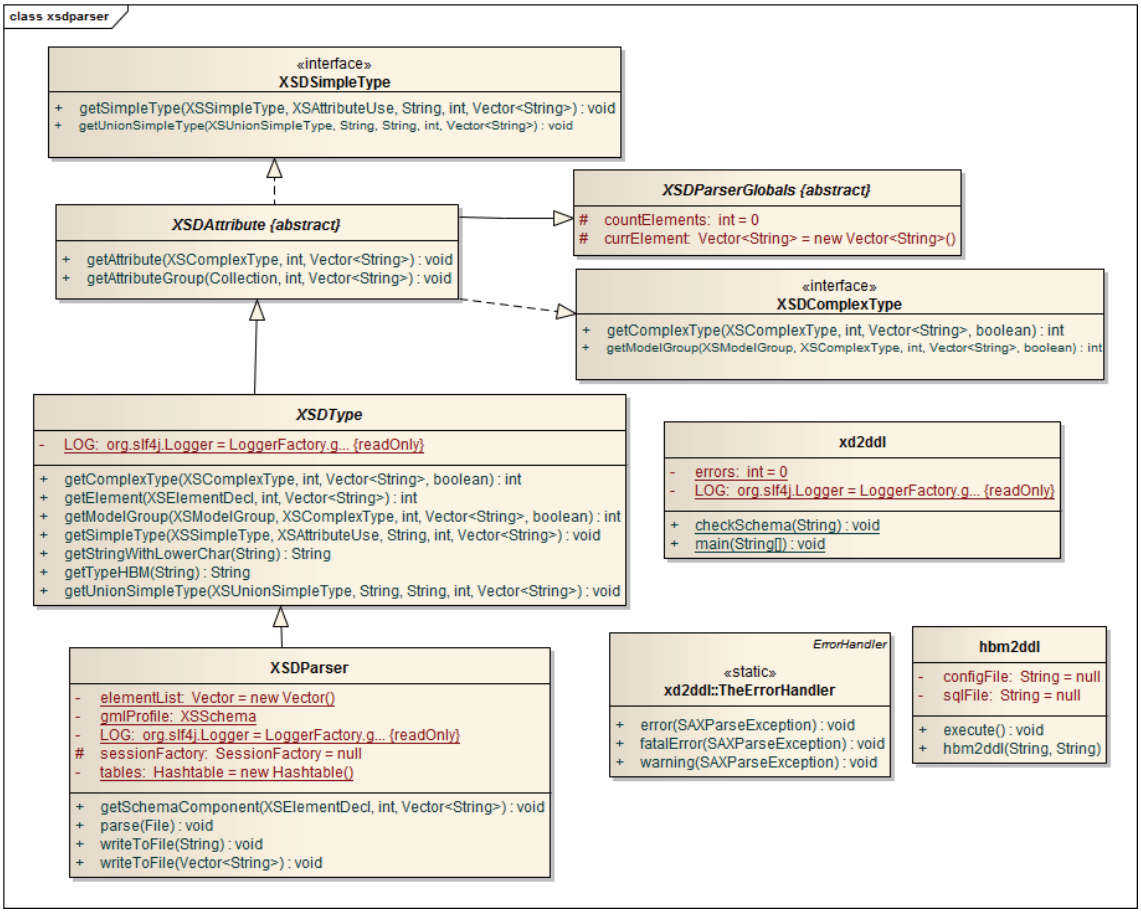


Abbildung 5.1.: xsd2ddl Klassenmodell

Für das Generieren von DLL wird das Java-Framework Hibernate (s. Abschn. 3.3.6) benutzt. Dieses nützliche ORM-Tool ist auf zahlreiche Java-Bibliotheken (u.a. log4j, dom4j und antlr) angewiesen und kann erst, nach Einbindung der Bibliotheken in den CLASSPATH der Java-Runtime-Umgebung, angewandt werden. Hibernate unterstützt die Anbindung von vielen Datenbanken (Oracle, PostgreSQL, mySQL) und kann auch mit PostGIS verwendet werden. Dazu muss die Java-Bibliothek des Projektes Hibernate-Spatial eingebunden werden. Nach Erstellung einer Hibernate-Mapping-Datei wird das Tool *hbm2ddl* aus den Hibernate-Tools von JBoss dazu genutzt werden um aus einer Hibernate-Mapping-Datei DDL, d.h. das Datenbankschema für PostgreSQL/PostGIS zu generieren.

In Anlehnung an das Klassenmodell des *XSOM* (s. Anhang C) ist der XML-Schema-Parser des Programms *xsd2ddl* in ähnlicher Art und Weise umgesetzt worden um einen modularen Aufbau zu erzielen ((s. Abb. 5.1)). In der Klasse *XSDParserGlobals* werden globale Variablen deklariert. Ein Zähler (*countElements*), der die Anzahl der verarbeiteten XML-Elemente aufzeichnet und ein *Vector* vom Datentyp *String* (*currElement*), der die Elemente zur Verarbeitung bereithält. Diese globalen Variablen werden an die Klassen weitervererbt bis zum *XSDParser*. Die Klasse *XSDAttribute* implementiert die Schnittstellen *XSDSimpleType* und *XSDComplexType* und vererbt ihre Methoden an die Klasse *XSDType*, wodurch die einfachen und komplexen Elemente und die Attribute von der Klasse *XSDType* verarbeitet werden. Gestartet wird das Programm über die Klasse *xsd2ddl*. Diese enthält den Schema-Validator und kann ein *XSDParser*-Objekt erzeugen. Dem *XSDParser* wird der Pfad zur XML-Schema-Datei übergeben. Nach einem erfolgreichen Parser-Vorgang wird eine Hibernate-Mapping-Datei mittels der Methode *writeToFile* geschrieben. Die Klasse *hbm2ddl* ist dafür vorgesehen das Hibernate-Tool *hbm2ddl* zu gebrauchen. Diese Klasse verlangt eine Hibernate-Konfiguration (hibernate.cfg.xml). Mittels der Methode *execute* wird das Tool über eine Hibernate-Session gestartet. Dieses kann entsprechend konfiguriert werden (s. Anhang, Abb. E.1).

In den Klassen *xsd2ddl*, *XSDParser* und *XSDType* wird zusätzlich der Log-Mechanismus des Projektes *Simple Logging Facade for Java* (SLF4J, vgl. [62]) für INFO oder DEBUG-Ausgaben verwendet. Die Art des Protokollierens wird in der Datei *log4j.properties* angepasst.

5.1.2. Benutzung

Das Programm verlangt den absoluten oder relativen Pfad zu einer XML-Schema-Datei als Übergabeparameter, wobei diese, weitere XML-Schema-Dateien beinhalten kann.

```
Aufruf: java -jar xsd2ddl <XML-Schema-Datei>
Beispielaufruf: java -jar xsd2ddl ./xplan/XPlanung-Operationen.xsd
```

1. Übergabe einer XML-Schema-Datei und des Formates an das Programm
2. Überprüfung, ob die XML-Schema-Datei existiert

3. Überprüfung des XML-Schemas auf Validität
4. Zergliederung der XML-Schema-Komponenten mit XSOM
5. Erzeugung von Hibernate-Mapping-Strukturen: *class*, *property*, *geometry-property*, *one-to-one* und *many-to-one*
6. Erzeugung und Speicherung der Hibernate-Mapping-Datei
7. Erzeugung der DDL mit *hbm2ddl*

Nachdem das Programm gestartet wurde, wird zunächst die Existenz der XML-Schema-Datei überprüft. Sollte sich bewahrheiten, dass die Datei nicht existiert, wird das Programm vom *ErrorHandler* abgebrochen. Im Anschluss wird das XML-Schema überprüft, ob es valide ist (vgl. Abschn. 2.5.1). Wenn das der Fall ist, wird das XML-Schema mittels XSOM-Methoden über die Klasse *XSDParser* zergliedert und in eine hbm-Datei geschrieben, andernfalls wird das Programm über den *ErrorHandler* abgebrochen. Das Hibernate-Tool *hbm2ddl* erzeugt die DDL, die je nach Einstellung in der Konfigurationsdatei (s. Anhang, Abb. E.1) weiter verarbeitet werden kann.

5.2. Tests und Auswertung

Verwendete Testmaschine

Prozessor	Intel Core 2 Duo P8400 2,26 GHz 2,27 GHz
Arbeitsspeicher (RAM)	4,00 GB
Betriebssystem	Windows Vista 32 Bit

Die Applikationen *Hyperjaxb*, *PostNAS*, *deegree-webservices* und der *xplan-manager* sind näher getestet worden. In den Tests ging es vordergründig darum, dass ein XML-Schema oder ein XML-Dokument ordnungsgemäß auf die relationale Datenbank abgebildet wurde.

Es hat sich gezeigt, dass PostNAS für einige GML-Anwendungsschema (NAS und INSPIRE Annex 1) benutzt werden kann. Das Format XPlanGML wird aber nicht ordnungsgemäß abgebildet. PostNAS kann die xlink-Referenzen (s.u.) des XPlanGML nicht auflösen. Dadurch fehlen Beziehungen zwischen den Plänen.

Beispiel aus einem XPlanGML-Dokument:

<xplan:gehoertZuPlan xlink:href="#GML_79eecd1b-f030-4c50-a359-732eb3d66992">

In XPlanGML-Dokumenten sind meistens unzählige solcher xlink-Referenzen enthalten und dürfen nicht vernachlässigt werden. Deswegen ist PostNAS für XPlanGML ungeeignet.

Hinzu kommt das XPlanGML-Dokumente zusätzliche Artefakte wie z.B. Bilder, Shape-Dateien oder PDFs enthalten.

Für die Speicherung von XPlanGML in den Versionen 2.0 und 3.0 ist der *xplan-manager* ein gutes Mittel zum Zweck. Zusätzliche Artefakte werden als *binary large object (blob)*, d.h. binär in der Datenbank gespeichert.

Mit der Beta-Version der *deegree-webservices* konnte der gesamte Anhang 1 von INSPIRE in relativ kurzer Zeit auf die Datenbank PostgreSQL abgebildet werden (s.u.).

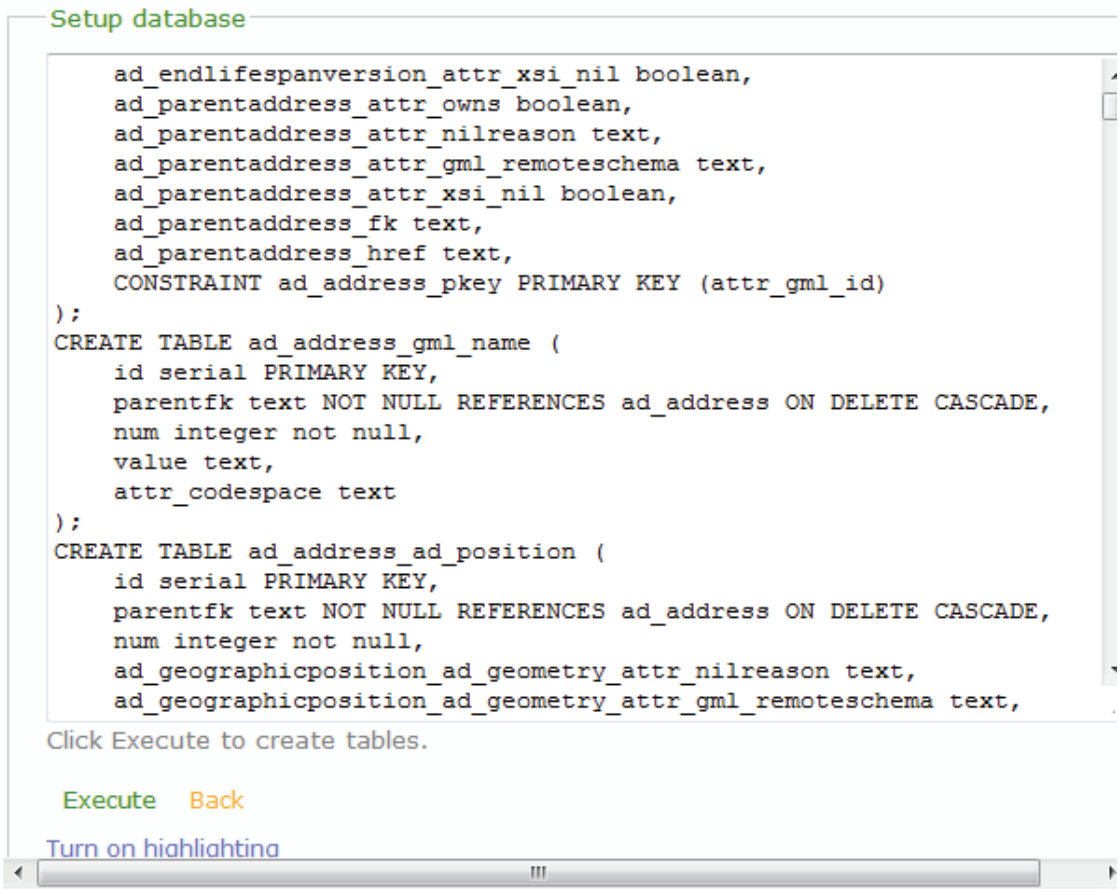


Abbildung 5.2.: DDL von INSPIRE Annex 1 in *deegree-webservices*

Nach der Ausführung von diesen 833 SQL-Statements konnten entsprechende GML-Dokumente eingelesen oder WFS-Transaktionen erfolgen (s. Abschn. 3.2.2).

Über die Anwendung Hyperjaxb konnten einfache und komplexe XML-Schemata in PostgreSQL-Datenbanken eingelesen werden. GML-Schemata konnten aber nicht erfolgreich importiert werden, da die bekannten Beziehungsprobleme und die Abbildung der Geometrie nicht gelöst werden konnte.

6. Ergebnis und Diskussion

Nach ausgiebiger Analyse verschiedener Werkzeuge für die Handhabung des objektrelationalen Mappings hat sich in den Tests herausgestellt, dass die Speicherung von XML in relationalen Datenbanken noch nicht selbstverständlich ist. Es konnten dennoch verschiedene GML-Anwendungsschemata über verschieden spezialisierte Programme eingelesen werden, darunter der gesamte Anhang 1 von INSPIRE, XPlanGML und NAS. Die *deegree-webservices* des deegree-Projektes zeigten einen guten Ansatz für die Bewältigung des ORM von GML. Von Seiten der deegree-Entwickler wurde aber betont, dass sie für ihre Projekte in der Regel noch manuell „mappen“ und die eigentliche Leistung der *deegree-webservices* (Beta-Version) in der Mapping-Sprache zu sehen ist und nicht im Wizard. Ein Problem ist, dass das Mapping-Verfahren dazu führt, dass die Tabellen- und Attributnamen sehr lang werden können, weil sich der XML-Baum in den Tabellennamen wiederspiegelt.

Das Projekt PostNAS zeichnet sich durch eine einfache Technik des Mappings aus. Diese Technik ist aber nicht für alle GML-Formate anwendbar, weil bspw. die Auflösung von „XPlanGML-xlink-Referenzen“ nicht unterstützt wird. XPlanungs-Daten sind da ein Spezialfall und können am besten über den *xplan-manager* von deegree verarbeitet werden.

Eigene Implementierungen brachten nur prototypische Resultate, weil durch den „ORM-Jungle“ das Wesentliche nicht sofort erkannt werden konnte. Insbesondere für GML-Anwendungsschemata und GML-Dokumente wurden dennoch geeignete Anwendungen recherchiert und getestet, die sehr gute Ergebnisse liefern (s. Tab. 6.1).

Format	**Anwendung**	
XML-Schema und -Dokument	Hyperjaxb / Hibernate	
	Anwendungschema	**Anwendung**
GML-Schema und -Dokument	INSPIRE und GeoSciML	deegree-webservices
	XPlanGML 2.0 und 3.0	xplan-manager
GML-Dokument	NAS	PostNAS

Tabelle 6.1.: Empfehlung für das ORM von XML und GML

Es hat sich gezeigt, dass Anwendungen zumeist auf die GML-Anwendungsschemata „zugeschnitten“ werden. Das deutet daraufhin, dass Interoperabilitätsprobleme nicht ausbleiben.

Die Ergebnisse zeigen, dass sich die Open-Source-Welt immer noch für relationale Datenbanken auch im GIS-Bereich einsetzt. Relationale Datenbanksysteme sind immer noch sehr beliebt, aber es stellt sich die Frage, wie objektorientierte Datenbanken mit XML umgehen können und, ob dadurch eventuell ein allgemein gültiges Konzept für das Mapping von GML hervorgebracht wird.

Der Layerdefinitionsgenerator für das WebGIS *kvwmap* ist nicht implementiert worden, weil die Lösung des Problems ORM insbesondere ORM von GML die gesamte Zeit beanspruchte.

7. Zusammenfassung und Ausblick

Die Analyse hat gezeigt, dass es eine Fülle von O/R-Mapping-Tools gibt, aber die wenigsten einen Anwendungsschema-Support haben. Nach eigenen Implementierungsversuchen wurden Erfolge verzeichnet im Hinblick auf das automatische Generieren von Datenbankstrukturen aus XML-Schemadateien. Wo andere Anwendungen scheiterten oder nur ein mangelhaftes Ergebnis lieferten, hatte sich die Anwendung *Hyperjaxb* zusammen mit Hibernate als generisches Tool für das objektrelationale Mapping von XML-Schemata und -Dokumenten herauskristallisiert.

Nach zahlreichen Recherchen zu GML wurde ein generischer Ansatz seitens des Projektes *deegree* gefunden. Das Projekt ist zwar noch sehr ausbaufähig, aber die Tests mit dem Wizard zeigten, dass dieser in der Lage ist, den gesamten Anhang 1 von INSPIRE ordnungsgemäß in eine PostgreSQL/PostGIS-Datenbank abzubilden. Daten konnten anschließend bequem per WFS-T oder über eine GML-Datei eingelesen werden. Das Projekt *PostNAS* hat sich insbesondere für das ORM von NAS-Daten bewährt.

Die Abbildung von XML und GML auf objektorientierte Datenbanken sollte näher untersucht werden, denn XML ist objektorientiert und der *object-relational impedance mismatch* ist nicht zu leugnen. Außerdem hat sich XML in die Informationsgesellschaft etabliert und ist dadurch nicht mehr wegzudenken. Interessant wäre zu untersuchen, wie insbesondere der weltweit führende Datenbankentwickler Oracle mit XML umgeht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV) (2011):* http://www.adv-online.de/icc/extdeu/broker.jsp?uMen=0a170f15-8e71-3c01-e1f3-351ec0023010 *(Stand Mai 2011).*

[2] *Ambler, S. W. (2002-2009): The Object-Relational Impedance Mismatch,* http://www.agiledata.org/essays/impedanceMismatch.html *(Stand Juni 2011).*

[3] *ArcGIS-INSPIRE-Lösung (2011):* http://www.esri.com/software/arcgis/arcgis-for-inspire/whats-included.html *(Stand Oktober 2011).*

[4] *Bittel (2010): Vorlesung OOP,* www-home.fh-konstanz.de/~bittel/prog2/Vorlesung/09OOP.pdf *(Stand August 2011).*

[5] *Bourret, R. (2005): DTD-Mapping,* http://www.rpbourret.com/xml/DTDToDatabase.htm *(Stand Mai 2011).*

[6] *Bourret, R. (2005): XML-DBMS,* http://www.rpbourret.com/xmldbms/index.htm *(Stand Juni 2011).*

[7] *Bourret, R. (2005): XML-DBMS-Alpha,* http://www.rpbourret.com/xmldbms/readme20.htm *(Stand April 2011).*

[8] *Brinkhoff, T. (2008): Geodatenbanksysteme in Theorie und Praxis, Wichmann Herbert, Auflage: 2, 2008.*

[9] *Caballero, N. (2001):* http://polaris.umuc.edu/~lpang/601/objectdb.ppt *(Stand August 2011).*

[10] *Chapple, M.: Database Keys,* http://databases.about.com/od/specificproducts/a/keys.htm *(Stand August 2011).*

[11] *Clark, J. (1997): Comparison of SGML and XML,* http://www.w3.org/TR/NOTE-sgml-xml-971215.html *(Stand April 2011).*

[12] *Coulon, X. / Brousseau, C. (2004): Hibernate simplifies inheritance mapping,* http://www.ibm.com/developerworks/java/library/j-hibernate/ *(Stand September 2011).*

[13] *deegree-webservices-3.1-rc4 (2011):* http://artefacts.deegree.org/libs-snapshots-local/org/deegree/deegree-webservices/3.1-rc4-SNAPSHOT/ *(Stand Oktober 2011).*

[14] *DIN EN ISO 19136 (2009): DINENISO19136: http://www.nabau.din.de/cmd?artid=116824897&contextid=nabau&bcrumblevel=1&subcommitteeid=54749703&level=tpl-art-detailansicht&committeeid=54738847&languageid=de (Stand Juni 2011).*

[15] *Eclipse (2011): http://www.eclipse.org/ (Stand September 2011).*

[16] *Faeskorn-Woyke, H. (2010): Datenbanken Online Lexikon, Fachhochschule Köln, http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Objekt-relationalesMapping (Stand Oktober 2011).*

[17] *FH-Wiesbaden: Datenabnksysteme, http://www.mi.fh-wiesbaden.de/~barth/dbs/vorl/DatenbanksystemePB13.pdf (Stand Juni 2011).*

[18] *Fiegler, M.: Objektorientierung in Oracle, http://www.ordix.de/ORDIXNews/1_2010/Datenbanken/objektorientierung_in_oracle.html (Stand August 2011).*

[19] *Galdos Systems Inc. (2011): INtune SDI Framework, http://www.galdosinc.com/technologies/products/intune (Stand September 2011).*

[20] *Gilsdorf, F.; Universität Trier: http://www.syssoft.uni-trier.de/systemsoftware/Download/Seminare/Middleware/middleware.9.book.html (Stand Juni 2011).*

[21] *GML4-Workshop (2011): http://external.opengis.org/twiki_public/GML/Gml4Workshop*

[22] *Hempel, T. (1997 - 2003): http://www.tinohempel.de/info/info/datenbank/integritaet.htm (Stand August 2011).*

[23] *Hibernate: Object relational mapping: http://www.hibernate.org/about/orm (Stand Juli 2011).*

[24] *Hibernate XML-Mapping (2004): http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/xml.html (Stand August 2011).*

[25] *Hibernate-Reference 3.6 (2004): http://docs.jboss.org/hibernate/core/3.6/reference/en-US/pdf/hibernate_reference.pdf, (Stand August 2011).*

[26] *HUANG, J.: http://web.eecs.utk.edu/~huangj/CS302S04/notes/oo-intro.html (Stand Juni 2011).*

[27] *HyperJAXB: http://confluence.highsource.org/display/HJ3/Home (Stand September 2011).*

[28] *International Data Corporation (IDC) (2011): http://www.idc.com/about/about.jsp (Stand Juni 2011).*

[29] *ISOTC 211 Geographic informationGeomatics (2007): ISO 19136-2007, Geographic information - Geography Markup Language (GML).*

[30] *ISO TC 211 Geographic informationGeomatics (2002): ISO 19101:2002, Geographic information - Reference model.*

[31] *ISO TC 211 Geographic informationGeomatics (2005): ISO 19109:2005, Geographic information - Rules for application schema.*

[32] *IT-Infothek (2003-2004): Wirtschaftsinformatik (Bachelor-Studiengang): Grundlagen der Kommunikationstechnik (4. Semester),* `http://www.it-infothek.de/fhtw/semester_4/grdlkt_4_04.html` *(Stand August 2011).*

[33] *JAXB (2003):* `http://www.oracle.com/technetwork/articles/javase/index-140168.html` *(Stand Oktober 2011).*

[34] *Karlsruher Institut für Technologie:* `http://www.iai.fzk.de/www-extern/index.php?id=679` *(Stand Mai 2011).*

[35] *Klettke, M. / Meyer, H. (2003): XML & Datenbanken - Konzepte, Sprachen und Systeme; dpunkt.verlag, 1. Auflage 2003, S.177.*

[36] *Kolbe, T. H.: Repräsentation von Geodaten mit GML,* `http://www.ikg.uni-bonn.de/vorlesungsarchiv/GIS_III/Folien/Pack-and-Go/unzip/gisIII.15.GML3-Teil1.ppt` *(Stand September 2011).*

[37] *Korduan, P. / Rahn, S. / Hentschel, M. (2011): GeoForum 2011 Konferenzbeitrag,* `http://kvwmap.geoinformatik.uni-rostock.de/GeoForum-2011.pps` *(Stand September 2011).*

[38] *Kresse, W. / Fadaie, K. (2004): ISO Standards for Geographic Information, Springer 2004*

[39] *Krüger, G. (1997): JAVA 1.1 lernen, Addison-Wesley Verlag, 1997,* `http://www.addison-wesley.de/Service/Krueger/kap07007.htm` *(Stand September 2011).*

[40] *Lahres, B. / Rayman, G. (2009): Objektorientierte Programmierung, Persistenz,* `http://openbook.galileocomputing.de/oop/oop_kapitel_06_001.htm` *(Stand Juli 2011).*

[41] *La-Mancha et al. (2009): Chapter I GML as Database: Present and Future,* `http://www.igi-global.com/viewtitlesample.aspx?id=20380` *(Stand August 2011).*

[42] *Libpq-Bibliothek (1996-011):* `http://www.postgresql.org/docs/9.1/static/libpq.html` *(Stand Oktober 2011).*

[43] *Milosavljevic et. al. ,CG & GIS Lab, Faculty of Electronic Engineering, University of Nis, Serbia,* `gislabweb.elfak.ni.ac.rs/alexm/publications/2010/Milosavljevic10.pdf` *(Stand August 2011).*

[44] *Grillenberger, R.:* `http://www.grillenberger.de/webdesign-glossar.htm` *(Stand September 2011).*

[45] *Mittermeier, L. (2003): XML und Datenbanken,* `http://www.heise.de/ix/artikel/Naiv-nativ-506250.html` *(Stand September 2011).*

[46] *NetBeans (2011):* `http://netbeans.org/` *(Stand September 2011).*

[47] *OGC (2010): OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option, Version 1.2.1, 4. August 2010,* `http://portal.opengeospatial.org/files/?artifact_id=25354` *(Stand September 2011).*

[48] *OGC (2007): Revision Notes for OpenGIS Implementation Specification: Geographic information - Geography Markup Language Version 3.2.1 (3.2.1),* `http://portal.opengeospatial.org/files/?artifact_id=26765` *(Stand September 2011).*

[49] *ORSÁG, J. (2006): Object Relational Mapping, Bratislava 2006*

[50] *PL/Java:* `http://pljava.projects.postgresql.org/` *(Stand Oktober 2011).*

[51] *Pohl, J. (1999-2000): Einführung in XML Teil III,* `http://www.jcpohl.de/technik/tut2xml.html` *(Stand Oktober 2011).*

[52] *PostgreSQL (1996-2011):* `http://www.postgresql.org/about/` *(Stand August 2011).*

[53] *PostGIS (2011):* `http://postgis.refractions.net/` *(Stand September 2011).*

[54] *PostNAS:* `http://www.wheregroup.com/de/postnas_distribution` *(Stand April 2011).*

[55] *PostNAS-trac:* `http://trac.wheregroup.com/PostNAS` *(Stand April 2011).*

[56] *Regis Con GmbH (2011):* `http://regiscon.com/index.php?ka=3&ska=42&\PHPSESSID=693918c1af6ee19b0cb928a442d9fee1` *(Stand Oktober 2011).*

[57] *Riekert, W.: SQL und relationale Algebra,* `http://v.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap7.htm` *(Stand Juni 2011).*

[58] *Schmidt, U. (2011): OOP mit Java: Objektorientierte Programmierung, Mehrfachvererbung,* `http://www.fh-wedel.de/~si/vorlesungen/java/OOP/Mehrfachvererbung.html` *(Stand August 2011).*

[59] *Schneider, M. (2008): Feature and object model, deegreeday2008,* `http://download.deegree.org/deegreeday2008/deegreeday2008_feature-and-object-model_Schneider_latlon.pdf` *(Stand September 2011).*

[60] *Segul Enterprises (2011): In C++, what's the diamond problem, and how can it be avoided?,* `http://www.programmerinterview.com/index.php/c-cplusplus/diamond-problem/` *(Stand August 2011).*

[61] *Shrestha, B. B. (2004): XML Database Technology and its use for GML, März 2004,* `www.itc.nl/library/Papers_2004/msc/gfm/shrestha.pdf`.

[62] *Simple Logging Facade for Java (2004-2011):* `http://www.slf4j.org/`, *(Stand Oktober 2011).*

[63] *Statistisches Bundesamt (2009): Informationsgesellschaft in Deutschland, Ausgabe 2009,* `http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Content/Statistiken/Informationsgesellschaft/InformationsgesellschaftDeutschland,property=file.pdf` *(Stand Juli 2011).*

[64] *Strobel, R. (2007): Die Bedeutung von Profilen für den Datenaustausch mittels Geography Markup Language, AGIS, Universität der Bundeswehr München,* `137.193.222.80/publikationen/download/strobel_bkg_2007.pdf`.

[65] *TU-Chemnitz: Laufzeitvergleiche von DOM und SAX,* `http://usr.informatik.tu-chemnitz.de/proseminare/xml04/doku/dom/frames/domsax.html` *(Stand Juni 2011).*

[66] *Türker, C. / Saake, G. (2006): Objektrelationale Datenbanken, dpunkt-verlag, Heidelberg, 2006*

[67] *UNI-Leipzig: DBSI-Buch,* `http://dbs.uni-leipzig.de/buecher/DBSI-Buch/HTML/kap13-2.html` *(Stand Juni 2011).*

[68] *Vonhoegen, H. (2009): Einstieg in XML: Grundlagen, Praxis, Referenz, Galileo Press, Bonn 2009, 5. Auflage.*

[69] *W3C (1999): XML Schema Requirements, W3C Note 15 February 1999,* `http://www.w3.org/TR/NOTE-xml-schema-req` *(Stand August 2011).*

[70] *W3C: XML-Schema-Type-Hierarchy,* `http://www.w3.org/TR/xmlschema-2/type-hierarchy.gif` *(Stand Juni 2011).*

[71] *W3C-SCHEMA-PART 0 (2004):* `http://www.w3.org/TR/xmlschema-0/` *(Stand Juni 2011).*

[72] *W3C-SCHEMA-PART 2 (2004):* `http://www.w3.org/TR/xmlschema-2/` *(Stand Juni 2011).*

[73] *W3C-XML (2008): Extensible Markup Language (XML) 1.0 (Fifth Edition),* `http://www.w3.org/TR/xml/` *(Stand April 2011).*

[74] *W3C-Schools: DOM,* `http://www.w3schools.com/dom/dom_nodetree.asp` *(Stand September 2011).*

[75] *W3C-Schools: DOM, books.xml,* `http://www.w3schools.com/dom/books.xml` *(Stand August 2011).*

[76] *W3C-School (1999-2011): XML-Schema,* `http://www.w3schools.com/schema/schema_why.asp` *(Stand Juni 2011).*

[77] *W3C-Schools (1999-2011): ComplexType,* `http://www.w3schools.com/schema/el_complexcontent.asp` *(Stand Oktober 2011).*

[78] *W3C-XML-Schema (2004): XML Schema Part 1: Structures Second Edition,* `http://www.w3.org/TR/xmlschema-1/` *(Stand Juni 2011).*

[79] *Wikipedia (2011): Datenbankmodellierung,* `http://de.wikipedia.org/wiki/Kardinalit%C3%A4t_%28Datenbankmodellierung%29` *(Stand August 2011).*

[80] *XML2DDL Berlios (2005):* `http://xml2ddl.berlios.de/` *(Stand April 2011).*

[81] *xplan-Manager (2010): Dokumentation,* `http://lkee-xplanung2.lat-lon.de/igeoportal/doc/manager_dokumentation_daten_spezifikation.pdf` *(Stand Oktober 2011).*

[82] *Xplan-node (2011):* `http://wiki.deegree.org/deegreeWiki/XplanNode` *(Stand August 2011).*

[83] *XML Schema Object Model (XSOM) (2008-2011):* `http://xsom.java.net/` *(Stand Juni 2011).*

[84] *XSD2DB (2005):* `http://xsd2db.sourceforge.net/` *(Stand Juni 2011).*

[85] *Zeidenstein, K. (2001): User-defined structured types and object views in DB2,* `http://www.ibm.com/developerworks/data/library/techarticle/zeidenstein/0108zeidenstein.html` *(Stand August 2011).*

[86] *Zentrum Elektronisches Publizieren in den Literaturwissenschaften (ZEPL) (2003):* `http://www.zepl.uni-muenchen.de/glossar.htm` *(Stand Oktober 2011).*

A. Simple Feature Modell

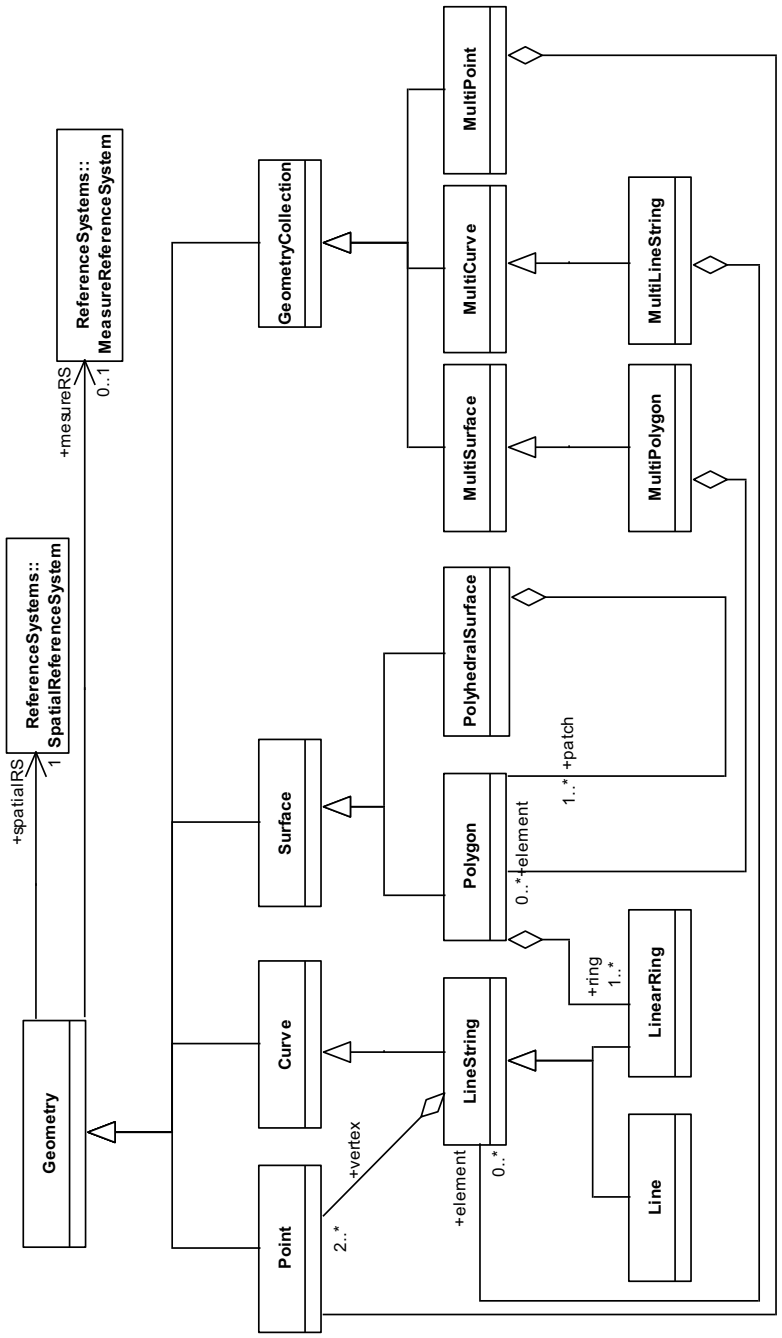


Abbildung A.1.: Simple Feature, Hierarchisches Geometrie-Modell [47]

B. XSLT Beispiel

```
1 <?xml version="1.0" encoding="UTF-8" ?><xsl:stylesheet version="1.0" xmlns:xsl="
    http://www.w3.org/1999/XSL/Transform"
2 xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xs="http://www.w3.org/2001/
    XMLSchema">
3 <xsl:output method="text" indent="no"/><xsl:strip-space elements="*"/>
4
5 <!-- Neue Zeile -->
6 <xsl:variable name="newline"><xsl:text>&#xa;</xsl:text></xsl:variable>
7 <!-- Einzug: Zwei Leerzeichen -->
8 <xsl:variable name="space"><xsl:text>  </xsl:text></xsl:variable>
9
10 <!-- Alle Elemente parsen -->
11 <xsl:template match="//xs:element[parent::xs:schema]">
12   <xsl:text>--DROP TABLE </xsl:text><xsl:value-of select="@name"/><xsl:value-of
       select="$newline"/>
13   <xsl:text>CREATE TABLE </xsl:text><xsl:value-of select="@name"/>
14   <xsl:text> (</xsl:text><xsl:value-of select="$newline"/>
15   <xsl:value-of select="$space"/><xsl:text>id_</xsl:text><xsl:value-of select="
       @name"/>
16   <xsl:text> serial PRIMARY KEY,</xsl:text><xsl:value-of select="$newline"/>
17   <xsl:variable name="currElement" select="@name"/>
18   <xsl:if test="@type">
19     <xsl:value-of select="$space"/><xsl:value-of select="$currElement"/>
20       <xsl:apply-templates select="@type | xs:simpleType/xs:restriction/@base"/>
21       <xsl:value-of select="$newline"/>
22   </xsl:if>
23   <!-- z.B. sequence oder choice, Kindelemente verarbeiten -->
24   <xsl:for-each select="//xs:element[ancestor::xs:element/@name=$currElement]">
25     <xsl:choose>
26       <!-- Fremdschluessel setzen -->
27       <xsl:when test="@ref">
28         <xsl:value-of select="$space"/><xsl:text>id_</xsl:text><xsl:value-of select
             ="@ref"/>
29           <xsl:text> INTEGER NOT NULL,</xsl:text><xsl:value-of select="$newline"/>
30           <xsl:value-of select="$space"/><xsl:text>CONSTRAINT </xsl:text><xsl:value
             -of select="@ref"/>
31           <xsl:text>_</xsl:text><xsl:value-of select="$currElement"/>
32           <xsl:text>_fkey FOREIGN KEY (id_</xsl:text><xsl:value-of select="@ref"/><
             xsl:text>) REFERENCES </xsl:text>
33           <xsl:value-of select="@ref"/><xsl:text>(id_</xsl:text><xsl:value-of
             select="@ref"/>
34           <xsl:text>) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION</
             xsl:text>
35           <xsl:choose>
36             <xsl:when test="position()!=last()"><xsl:text>,</xsl:text><xsl:value-of
                 select="$newline"/></xsl:when>
37             <xsl:otherwise><xsl:value-of select="$newline"/></xsl:otherwise>
38           </xsl:choose>
39       </xsl:when>
40       <xsl:otherwise>
41       <xsl:value-of select="$space"/><xsl:value-of select="@name"/>
```

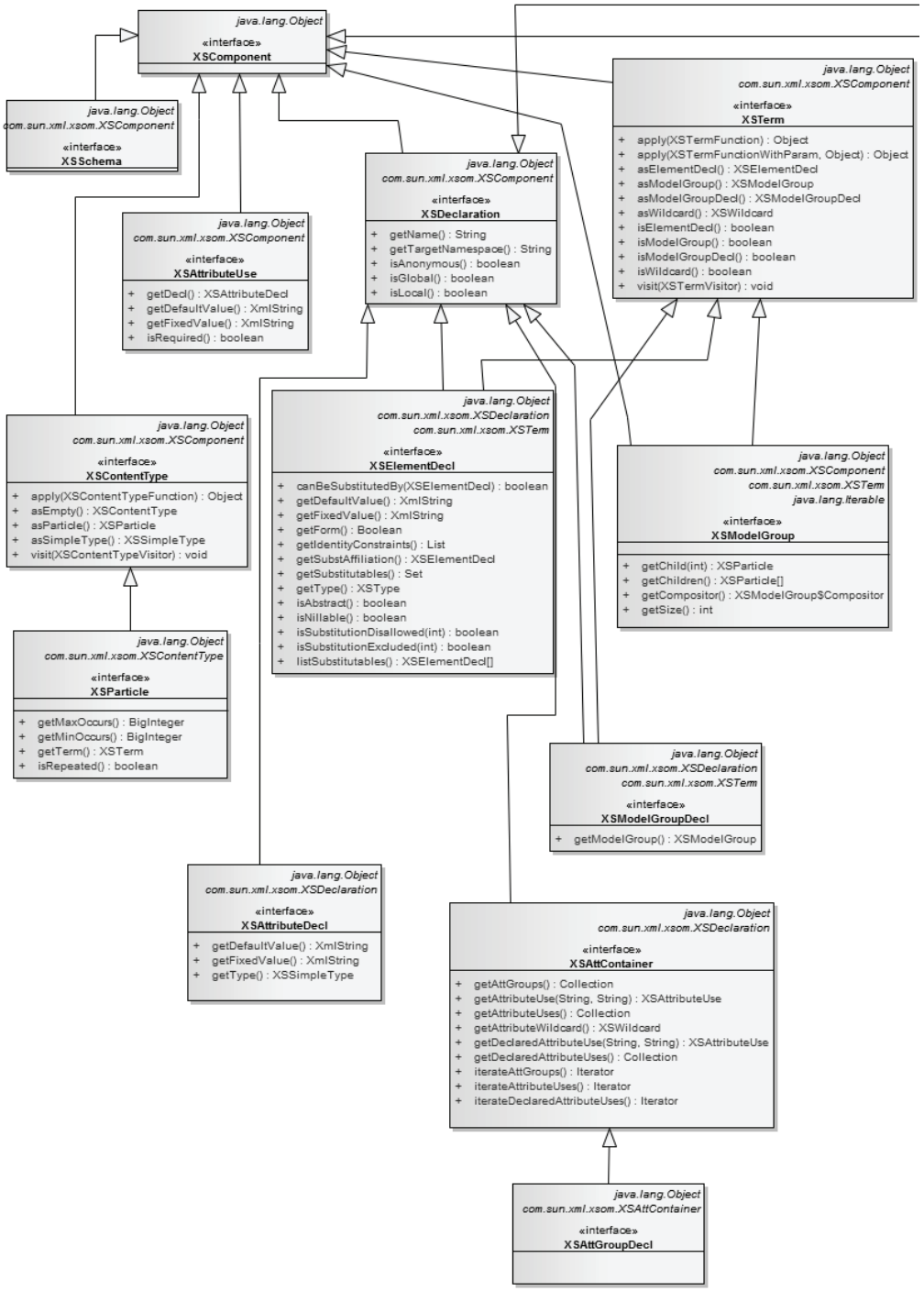
```
42          <xsl:apply-templates select="@type | xs:simpleType/xs:restriction/@base"/>
43          <xsl:choose>
44            <xsl:when test="position()=last()"><xsl:value-of select="$newline"/></
                xsl:when>
45            <xsl:otherwise><xsl:text>,</xsl:text><xsl:value-of select="$newline"/></
                xsl:otherwise>
46          </xsl:choose>
47        </xsl:otherwise>
48      </xsl:choose>
49    </xsl:for-each>
50    <xs:text>);</xs:text><xsl:value-of select="$newline"/><xsl:value-of select="$
        newline"/>
51 </xsl:template>
52
53 <!-- XSD-Datentypen in Datenbank-Datentypen umwandeln -->
54 <xsl:template match="@type | xs:simpleType/xs:restriction/@base">
55    <xsl:variable name="val" select="."/>
56    <xsl:text> </xsl:text>
57     <xsl:choose>
58     <xsl:when test="$val = 'xs:double'"><xsl:text>REAL</xsl:text></xsl:when>
59     <xsl:when test="$val = 'xs:integer'"><xsl:text>INTEGER</xsl:text></xsl:when>
60       <xsl:when test="$val = 'xs:string'"><xsl:text>VARCHAR(255)</xsl:text></
            xsl:when>
61       <xsl:when test="$val = 'xs:dateTime'"><xsl:text>DATE</xsl:text></xsl:when>
62     </xsl:choose>
63   </xsl:template>
64  </xsl:stylesheet>
```

Listing B.1: XSLT-Beispiel

```
1  --DROP TABLE Angebot
2  CREATE TABLE Angebot (
3    id_Angebot serial PRIMARY KEY,
4    id_Preis INTEGER NOT NULL,
5    CONSTRAINT Preis_Angebot_fkey FOREIGN KEY (id_Preis) REFERENCES Preis(id_Preis)
         MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION,
6    id_Artikel INTEGER NOT NULL,
7    CONSTRAINT Artikel_Angebot_fkey FOREIGN KEY (id_Artikel) REFERENCES Artikel(
         id_Artikel) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION
8  );
9
10 --DROP TABLE Artikel
11 CREATE TABLE Artikel (
12   id_Artikel serial PRIMARY KEY,
13   Name VARCHAR(255),
14   Hersteller VARCHAR(255),
15   id_Preis INTEGER NOT NULL,
16   CONSTRAINT Preis_Artikel_fkey FOREIGN KEY (id_Preis) REFERENCES Preis(id_Preis)
         MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION
17 );
18
19 --DROP TABLE Preis
20 CREATE TABLE Preis (
21   id_Preis serial PRIMARY KEY,
22   Preis REAL
23 );
```

Listing B.2: DDL-Beispiel

C. XSOM-Modell



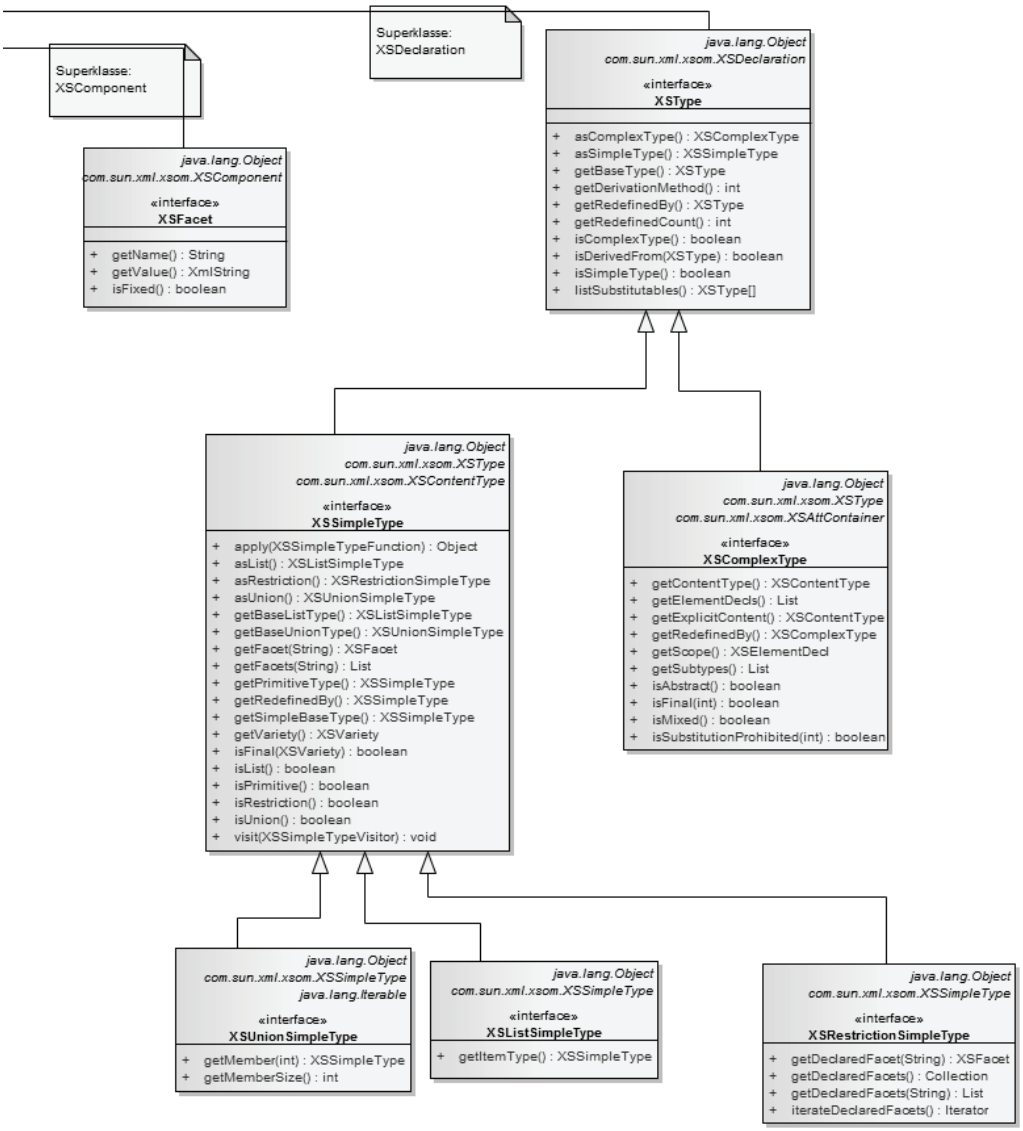
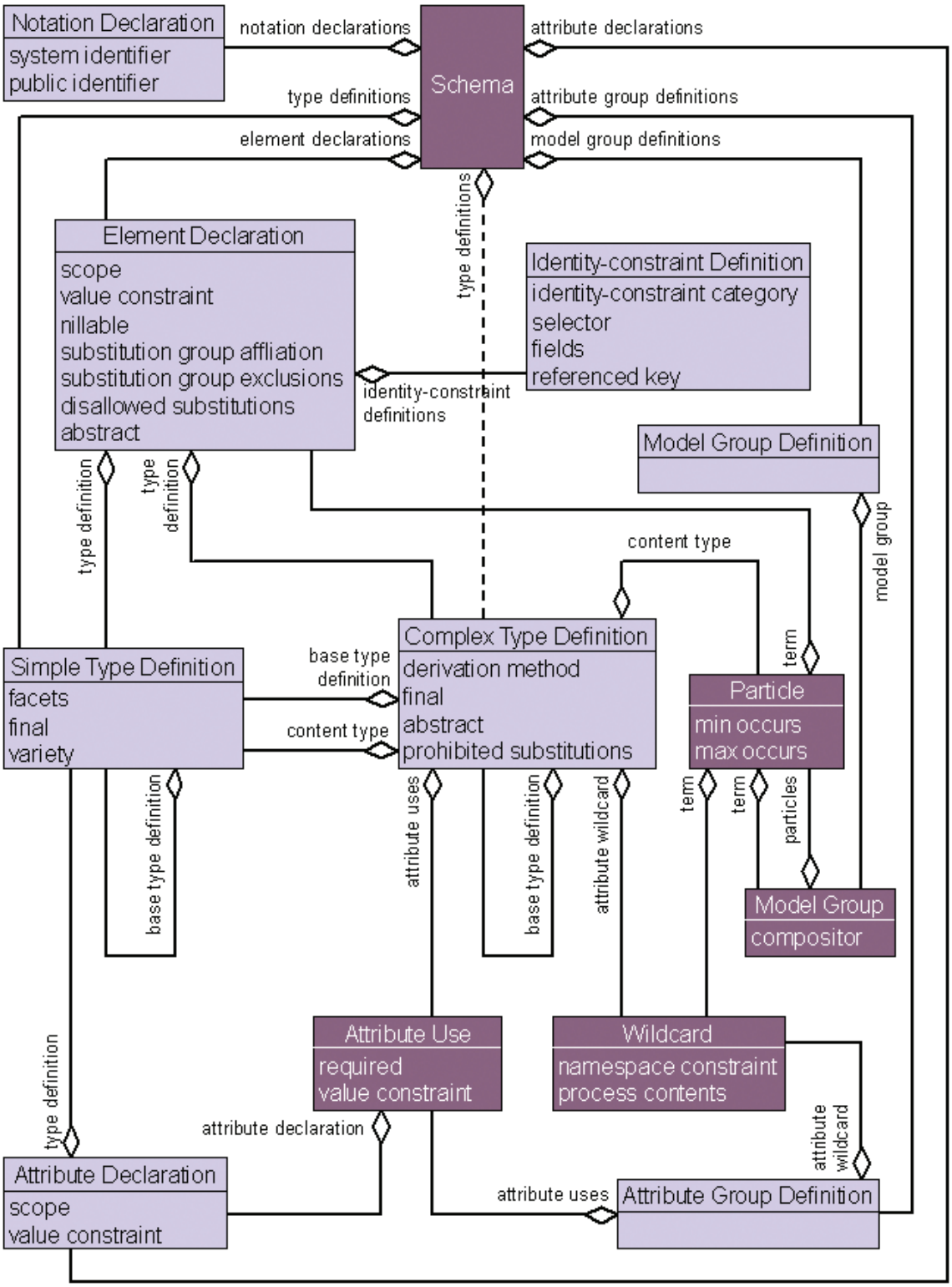


Abbildung C.1.: XSOM-Interface-Model

D. XML Schema Component Data Model



E. Hibernate-Konfiguration

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
      <!-- Verbindung zur Datenbank -->
      <property name="connection.driver_class">
       org.postgresql.Driver
    </property>
      <property name="connection.url">
    jdbc:postgresql://localhost/test
    </property>
    <property name="connection.port">5432</property>
    <property name="connection.username">postgres</property>
    <property name="connection.password">postgres</property>
    <property name="connection.pool_size">1</property>
    <!-- Postgis -->
    <property name="dialect">
       org.hibernatespatial.postgis.PostgisDialect
      </property>
    <property name="current_session_context_class">thread</property>
    <property name="cache.provider_class">
       org.hibernate.cache.NoCacheProvider
      </property>
    <!-- SQL anzeigen -->
    <property name="show_sql">true</property>
    <!-- schoen formatieren -->
    <property name="format_sql">true</property>
    <!-- direktes Einlesen -->
    <property name="hbm2ddl.auto">create</property>
    <!-- Hibernate-Mapping-Datei -->
    <mapping resource="XPlanung-Operationen.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Listing E.1: hibernate.cfg.xml

F. Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich diese Masterarbeit selbstständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Neubrandenburg, den 28. Oktober 2011

Matthias Pramme