Geodateninfrastrukturservice Rostock
Dr.-Ing. Peter Korduan

Vervollständigung des PostNAS-Schemas und die Anpassung des NAS-Imports

Erstellt von: Peter Korduan, GDI-Sevice
letzte Änderung am: 19.10.2016

Inhaltsverzeichnis

1 Einleitung

XMI Dateien sind XML-Repräsentationen von UML-Modellen. Manchmal braucht man die UML Modellelemente, besonders die Klassen, seine Attribute, die Assoziationen und Generalisierungen, in einer Datenbank-Tabellenstruktur, z.B. zur Ableitung eines Datenbankmodells zur Speicherung von ALKIS-Daten.

Das entwickelte Programmpaket xmi2db liest eine XMI Datei und schreibt die UML Dinge in die Datenbankstrktur, welche sich an UML Strukturen orientiert. Es gibt Tabellen für Klassen, Attribute, Generalisierungen, Datentypen, Stereotypen usw.

Im nächsten Schritt kann mit der Funktion db2classes ein relationales Datenbankschema erzeugt werden, welches für jede einzelne Klasse eine separate Tabelle erzeugt, mit Attributen, die zur Klasse passen. Die Tabellendefinition berücksichtigt die Generalisierung von UML-Klassen und die Vererbung. Multiplizität wird durch Definition der Attribute als Arrays berücksichtigt. Die Assoziationen werden verbunden durch gml_id Attribute vom Typ uuid. Die Funktion db2ogr erzeugt ein relationales Databankmodell ohne komplexe Datentypen. Dieses Schema kann für den Import von GML-Dateien mit ogr2ogr verwendet werden.

Der xmi2db converter fokusiert sich auf GML-Anwendungsschemas wie die für INSPIRE, das AAA-Modell oder XPlanung-Schema. Der Type UNION wird in geometry umgesetzt und die PostGIS Erweiterung für die Datenbank ist erforderlich. In den folgenden Abschnitten werden die Arbeitsschritte im Projekt beschrieben.

Im ersten Arbeitsschritt musste das vorhandene AAA-UML-Modell in ein Implementierungsmodell überführt werden. Diese Ableitung wird im Abschnitt 2 beschrieben. Dieses Implementierungsmodell wird als XMI-Datei aus Enterprise Architekt exportiert und mit dem entwickelten Programmpaket xmi2db in eine PostgreSQL-Datenbank eingelesen, siehe Abschnitt 3.

Aus den eingelesenen Informationen über das UML-Modell kann nun ein Schema erzeugt werden in dem jeder FeatureType, jede Aufzählung und jede Codeliste einer Tabelle entspricht sowie Datentypen und Aufzählungen Postgres Datentypen sind. Die Programmfunktion db2classes, die auch nutzerspezifische

Spalten und Beziehungstabellen anlegt wird im Abschnitt 4 beschrieben.

Der wichtigste und wohl auch komplexeste Arbeitsschritt ist das „flach“ machen des komplexen Schemas. Das Werkzeug zum Einlesen von NAS-Dateien ogr2ogr verlangt ein Datenmodell in dem Attribute von Datentypen auf die Tabelle, die sie benutzt übertragen werden. Dabei mussten auch mehrfach verschachtelte Datentypen berücksichtigt werden. Das „flach“ machen erfordert auch die Umbenennung einiger Attribute. Dieser Arbeitsschritt wird mit dem Programm db2ogr umgesetzt und ist in Abschnitt 5 beschrieben.

Der folgende Abschnitt 6 enthält eine Schritt für Schritt Anleitung zum Einlesen von NAS-Dateien in das neue vervollständigte ogr Schema.

Das letzte Kapitel 7 beschreibt die Abstimmungen und Zusammenarbeit mit der PostNAS Anwendergruppe.

2 Ableitung des Implementierungsmodells

2.1 Hintergrund

„Das AAA-Anwendungsschema verwendet einige Konstruktionen in UML, die in den Abbildungsregeln von ISO 19136 Annex E und ISO/TS 19139 nicht unterstützt werden. Daher erfolgt eine skriptgestützte Umsetzung des konzeptuellen AAAAnwendungsschemas in UML in ein Implementierungsschema“ [GeoInfoDok] Abschnitt 4.4.2 Das Skript mit dem Namen Shape Change nimmt die im Abschnitt 4.4.2 beschriebenen Änderungen am UML Modell vor. Dazu gehören auszugsweise folgende Punkte:

- Reduktion von multipler Vererbung
- nicht navigierbare Assoziationsrollen werden navigierbar durch den Zusatz von inversZu und auf Kardianlität von 0 gesetzt.
- Modellelemente, die Inhalte besitzen, die nicht in die NAS umgesetzt werden, werden bei der Ableitung des Implementierungsmodells für den Datenaustausch entfernt.
- Spezifische Anpassungen, die in der NAS anders umgesetzt werden sollen als im Modell vorgesehen
- Löschen einiger Klassen

2.2 Installation von Shapechange

Die Datei SSJavaCom.dll wird im System-Pfad abgelegt: (<Windows Verzeichnis>/System32 auf einem 32-Bit-System, <Windows Verzeichnis>/SysWOW64 auf einem 64-Bit-System). Quelle dieser Datei ist: <EA-Installationsverzeichnis>/Java API/

AAATools-1.0.2.zip von hier herunterladen:
http://shapechange.net/resources/dist/de/interactive_instruments/ShapeChange/AAATools/1.0.2/AAATools-1.0.2.zip und dann entpacken.

2.3 Konfiguration

Direkt im Hauptordner der AAATools muss die Datei nas.bat, also die Datei mit der die Konvertierung durchgeführt wird, geändert werden. Sie zeigt auf die Konfiguration „NAS-7.0.2.xml“. Will man NAS 6.0.1 konvertieren muss deshalb in dieser Datei auf die Datei „NAS-6.0.1.xml“ verwiesen werden. Sie hat dann folgenden Inhalt:

```
java -jar AAATools-1.0.2.jar -c "Konfigurationen/NAS-6.0.1.xml"
```

Die weitere Konfiguration erfolgt im im Verzeichnis „Konfigurationen“ der AAATools in der Datei „NAS-6.0.1.xml“. Hier muss in Zeile 5 nur die Input-Datei geändert werden (vorkonfiguriert ist „AAA-6.0.1-Kopie.eap“). Die Zeile sieht richtig konfiguriert dann so aus:

```
<parameter name="inputFile" value="AAA-6.0.1.eap"/>
```

2.4 Durchführung

Einfach die nas.bat starten und warten bis die Konvertierung abgeschlossen ist.

Achtung: Das normale Modell geht dabei verloren! Denn die konvertierte Datei heißt auch AAA-6.0.1.eap und damit ist das Original überschrieben.

3 UML-Modell in Datenbank einlesen (xmi2db)

Zur Umsetzung des Werkzeuges wurde sich für eine Web-Entwicklung in PHP entschieden. Auch wenn es nicht der Fokus der Entwicklung war, wäre es somit auch möglich, beliebige XMI Dateien auf einen Server hochzuladen und am Ende ein SQL-Skript zu erhalten, mit dem die eigene Datenbank gefüllt werden kann. In dieser Entwicklung werden die XMI-Dateien jedoch direkt in einem Verzeichnis auf dem Server abgelegt, das von der rudimentären Oberfläche abgefragt wird und alle dort abgelegten XMI-Dateien auflistet (siehe Abbildung). Außerdem wird das Modell in einem auswählbaren oder neu anzulegenden Schema in einer Datenbank auf dem Server gespeichert (Punkt „Schemaauswahl/-eingabe“ in der GUI). Kann das angegebene Schema nicht in der Datenbank gefunden werden, wird ein neues Schema angelegt und die Tabellenstruktur erzeugt. Darüber hinaus kann angegeben werden, ob man ein spezifisches Paket *package* und dessen Unterpakete transformieren möchte, dann füllt man das Feld „BasePackageauswahl/-eingabe“ aus oder ob alle Pakete in die Datenbank geladen werden sollen, dann lässt man das Feld leer. Die Option *truncate* sorgt bei einem bestehenden Schema dafür, dass alle Tabellen zunächst geleert werden, bevor sie neu befüllt werden.
Todo: Auf AAA anpassen und präzisieren
Die letzte Option „Argo Export mit ISO19136 Profil“ kann für den Fall aktiviert werden, dass es sich um einen Export aus ArgoUML unter Nutzung des ISO19136 Profils handelt. Hierbei sind die Datentypen und Stereotypen nur durch eine URI angegeben und sind somit nicht entschlüsselbar. Eine Aktivierung der Option füllt die Tabellen *Datatypes* und *Stereotypes*, womit beispielsweise klar wird, dass ein Element, dem ein Datentyp mit der URI http://argouml.org/user-profiles/ISO19136_Profile.xmi#-117-30-110-24-3c98ba4d:11d6bf2b1c7:-8000:0000000000001158 zugewiesen wurde, ein „CharacterString“ ist.

xmi2db

Führe als erstes die Funktion xmi2db "Fülle DB mit XMI Inhalten" aus um die UML-Elemente nach Postgres einzulesen.

Die Funktion erzeugt das Datenbankschema zur Speicherung der UML-Elemente und liest alle Klassen, Attribute, Beziehungen, Generalisierungen und Assoziationen aus der XMI-Datei aus und trägt sie in Tabellen ein.

* Wähle die XMI Datei aus dem Ordner xmis aus,

* Wähle ein Schemaname, z.b. aaa_uml

* Wähle den Namen des Basis Paketes des UML models aus.

* Klick auf "Fülle DB mit XMI Inhalten"

Die Schemanamen können auch in conf/database_config.php vordefiniert werden.

4 UML-Schema in Klassenschema überführen (db2classes)

Diese Funktion erzeugt den SQL-Code eines Datenbankschemas, welches für jede UML-Klasse eine Tabelle hat, für jeden Datentyp einen Postgrestyp und für jede Aufzählung einen Enumerationstyp und je eine Schlüsseltabelle für eine Enumeration und eine Codeliste.

* Wähle den Namen des Schemas aus in das die XMI eingelesen wurde.

* Wähle den Namen des Ausgabeschemas aus.

5 UML-Schema in OGR Schema überführen (db2ogr)

Diese Funktion erzeugt den SQL-Code eines flachen Datenbankschemas, welches für jede UML-Klasse eine Tabelle hat. Die Attribute sind jedoch nicht mit komplexen Datentypen versehen, sondern die Attribute der Datentypen sind als Attribute der Tabelle übernommen.

Da nicht alle im Implementierungschema verwendeten Datentypen im UML-Modell vorhanden sind, wurden diese explizit an Hand der Vorgaben in ISO 19136 erzeugt. Folgende Typen wurden angelegt:

siehe Abschnitt „Create DataTypes not definend in UML-Model“ in db2classes.php

SC_CRS

- scope character varying

doubleList

- list text

measure

- value integer

Des Weitere wurden die Typen wfs:transaction und wfs:query aus der Web Feature Specifikation mit einfachen Typen abgebildet.

Transaction

- content text

Query

- url character varying

Folgende Datentypen werden auf Grund ihrer Komplexität zunächst als text festgelegt, siehe Funktion get_database_type in classes/attribute.php

- Li_Lineage

Folgende komplexere Geometrietypen werden in einfachere umgewandelt.

- gm_compositecurve => MultiLinestring
- gm_solid => MultiPolygon
- gm_compositesolid => MultiPolygon
- gm_triangulatedsurface => MultiPolygon

Um doppelte Namen zu vermeiden werden einige Attribute umbenannt. Eine Liste der Umbenennungen kann mit der URL http://yourserver.de/xmi2db/listings/umbenennungsliste.php erzeugt werden. Zur Ausführung wird ein Schema ausgewählt in das die XMI eingelesen sind und ein Name für das Ausgabeschema angegeben.

Um einen tieferen Einblick zu erhalten was alles abgefragt wird um die Schmata zu erzeugen, kann der Parameter loglevel=1 mit angegeben werden. z.B.

```
http://meinserver.de/xmi2db/converter/db2classes.php?
umlSchema=aaa_uml&gmlSchema=aaa_gml&loglevel=1
```

Filter

Das Schema, welches mit db2ogr erzeugt wird, kann durch einen Filter beschränkt werden. Dazu dient eine Filterdatei im JSON Format, dessen Name in conf/database_conf.php im Parameter FILTER_FILE eingestellt werden kann. Die Beispieldatei conf/filter_sample.json enthält folgende Filter.

```
{
      "AA_Modellart": {
            "attribute": {
                  "sonstigesModell": 0
            }
      },
```

```
      "AA_Objekt": {
            "beziehungen": {
                  "istTeilVon": 0
            }
      },
      "AX_Netzknoten": 0,
      "AX_Bauwerksfunktion_Leitung": 0
}
```

Im Element AA_Modellart wird das Attribut sonstigesModell ausgeschlossen. Im Element AA_Objekt wird die Beziehung istTeilVon ausgeschlossen. Zusätzlich wird das Elemente AX_Netzknoten und die Aufzählungsklasse AX_Bauwerksfunktion_Leitung vollständig weggelassen. Es können mehrere Attribute und Beziehungen getrennt durch Komma angegeben werden. Die Zahl hinter dem : hat noch nichts zu sagen und sollte mit 0 angegeben werden.

6 Schritt für Schritt Anleitung

6.1 Installation xmi2db

Um das ogr-Schema zu erhalten, in welches NAS-Daten eingelesen werden sollen, wird keine Software benötigt. Das vorgefertigte Schema und das Umbenennungsskript stehen zum Download unter

http://gdi-service.de/xmi2db/converter/db2ogr.php

http://gdi-service.de/xmi2db/converter/rename_nas.rb

zur Verfügung. Wer jedoch ein angepasstes Schema mit eigenen Einstellungen erzeugen möchte kann die ersteltte Software wie folgt installieren.

1. Clone das Projekt in das eigene Verzeichnis.

```
git clone https://github.com/pkorduan/xmi2db.git
```

2. Erzeuge und editiere die Datei database_config.php

```
cp conf/database_conf_sample.php conf/database_conf.php
```

3. Passe Datenbankzugang an: PG_HOST, PG_USER, PG_PASSWORD, PG_DBNAME

4. Erzeuge eine Datenbank, die $PG_USER gehört und installiere die Erweiterung PostGIS

5. Lege die zu importierende XMI-Datei im Unterordner xmis ab.

6. Öffne den Link um auf die Konvertierungsoberfläche zu kommen. http://yourserver.de/xmi2db/

6.2 Installation libxml-ruby

Um NAS-Dateien in das neue flache Schema, welches bei db2ogr herauskommt einlesen zu können, müssen einige XML-Elemente umbenannt werden. Dazu wurde das Ruby-Program rename_nas.rb geschrieben, welches sich im Verzeichnis converter befindet. Die Ausführung unter Debian erfordert die Installation von libxml-ruby. Dieses Debianpaket wird wie folgt installiert:

```
apt-get update && apt-get install ruby-libxml
```

6.3 Einlesen vorbereiten

Zum Einlesen von NAS-Dateien in Postgres benötigt man in der Datenbank ein aufbereitet Schema im folgenden "aaa_ogr" genannt und eine aufbereitete NAS-Datei im folgenden "renamed_nas.xml" genannt.

6.3.1 Erstellung des Schemas "aaa_ogr"

Ein vollständiges Schema kann unter

http://gdi-service.de/xmi2db/converter/db2ogr

heruntergeladen werden, z.B. in der Datei aaa_ogr_schema.sql ablegen.

Länderspezifische Schemata lassen sich mit dem Zusatz filter= mv,rp oder sl erzeugen. z.B.

http://gdi-service.de/xmi2db/converter/db2ogr?filter=mv. Siehe Punkt "Filter" oben, um zu erfahren was gefiltert wird und wie er funktioniert.

Den SQL-Text in aaa_ogr_schema.sql in einer Datenbank in einem SQL-Client ausführen z.B. pgAdmin3 oder psql ausführen. Die Befehle zum Anlegen der Datenbank lauten wie folgt:

```
CREATE DATABASE "mypgdatabase";
CREATE EXTENSION postgis;
```

Befehl zum Ausführen der SQL-Datei aaa_ogr_schema.sql im Console-Client psql:

```
psql -U mydbuser -f aaa_ogr_schema.sql mydbname
```

siehe psql --help für mehr Informationen.

6.3.2 NAS-Datei aufbereiten

Jede NAS-Datei, die in das Schema aaa_ogr eingelesen werden soll, muss voher mit dem Script "rename_nas.rb" aufbereitet werden. Zur Installation von ruby siehe Abschnitt 6.1.

Die Umbenennung von Elementen in einer NAS-Datei "eingabedatei.xml" wird wie folgt aufgerufen:

```
ruby rename_nas.rb eingabedatei.xml [ausgabedatei.xml]
```

6.4 Einlesen

6.4.1 Eine einzelne NAS-Datei einlesen

Eine einzelne aufbereitete NAS-Datei "renamed_nas.xml" wird wie folgt mit ogr2ogr in das Schema "aaa_ogr" eingelesen.

```
ogr2ogr -f "PostgreSQL" --config PG_USE_COPY NO -nlt CONVERT_TO_LINEAR
-append PG:"dbname=mydbname active_schema=aaa_ogr user=mydbuser host=myhost
port=5432" -a_srs EPSG:25833 renamed_nas.xml
```

Im Osten Deutschlands wie Mecklenburg-Vorpommern nutze 25833 sonst 25832

Siehe http://gdal.org/drv_nas.html für mehr Informationen zur Benutzung von ogr2ogr

6.4.2 Automatisierung des Einlesens von Massendaten

NAS-Dateien, die im nutzer- oder stichtagsbezogenem Abgabeverfahren (NBA) von AAA-Softwaresystemen erzeugt werden, liegen in der Regel in Form von gepackten und komprimierten Archiven vor, z.B. NBA_Grundausstattung_2015-02-11.zip Unter Linux lassen sich solche Archive wie folgt entpacken:

```
unzip NBA_Grundausstattung_2015-02-11.zip
```

Es entstehen viele Dateien z.B.

```
NBA_Grundausstattung_001_081_2015-02-11.xml.gz
NBA_Grundausstattung_002_081_2015-02-11.xml.gz
...
NBA_Grundausstattung_081_081_2015-02-11.xml.gz
```

Diese Dateien wiederum lassen sich wie folgt entpacken und in einer Schleife verarbeiten.

```
gunzip *.xml.gz
for NAS_FILE in *.xml
do
  ruby rename_nas.rb $NAS_FILE renamed_nas.xml
   ogr2ogr -f "PostgreSQL" --config PG_USE_COPY NO -nlt CONVERT_TO_LINEAR -append PG:"dbname=mydbname active_schema=aaa_ogr user=mydbuser host=myhost port=5432" -a_srs EPSG:25833 renamed_nas.xml
done
```

In der Schleife der Abarbeitung ist jedoch noch zu berücksichtigen, dass die erste Datei Metadaten enthält und ignoriert werden kann und Fehler abgefangen werden müssen.

Ein Vorschlag für ein Bash-Skript für Linux, welches die Metadaten und Fehlerbehandlung berücksichtigt in Log-Dateien protokolliert und abgearbeitete Dateien in einen Archivordner schreibt, findet sich in der Datei converter/import_nas.sh Die Datei muss im Modus "Ausführbar" sein

```
chmod a+x import_nas.sh
```

Passen Sie vor dem Ausführen der Datei mit

```
./import_nas.sh
```

die folgenden Parameter an.

```
DATA_PATH="/pfad/zu/den/nas/dateien"
OGR_PATH="/pfad/zu/ogr2ogr/bin/verzeichnis"
ERROR_LOG=/pfad/fuer/logfiles/mylogfile.log
```

7 Abstimmung mit PostNAS

Auf der FOSSGIS 2016 am 4.7.2016 in Salzburg fand ein Anwendertreffen von PostNAS statt. Auf dem Anwendertreffen hat Peter Korduan, Geschäftsführer der Firma GDI-Service Rostock das Vorhaben zur Erweiterung des OGR Datenmodells vorgestellt.

Zunächst wurde die Problemstellung mit dem unvollständigen OGR-Modell sowie den doppelt vorkommenden und abgeschnittenen Attributnamen dargelegt. Die daraus resultierende Aufgabenstellung wurde vom Landkreis Vorpommern-Rügen, dem Landesamt für Vermessung und Geobasisinformation Rheinland-Pfalz und dem Landeamt für Vermessung, Geoinformation und Landentwicklung des Saarlandes beauftragt. Sie umfasst folgende Punkte:

1. Umwandlung des AAA-UML-Modells in das AAA-Implementierungsschemas mit ShapeChange
2. Einlesen der aus Enterprise Architekt exportierten xmi-Datei in eine Postgres-Datenbank
3. Transformation dieses Meta-Modells in ein Schema, welches die GML-Klassen abbildet
4. Ableitung des Datenschemas für OGR durch „flach machen“ des objektorientierten Klassenschemas und Anwendung von generischen Umbenennungsregeln sowie länder-, bzw. anwendungsspezifischer Filterung
5. Erstellung eines Scripts zur Umbenennung von verschachtelten GML-Elementen in einzulesenden NAS-Dateien

Zum Zeitpunkt des Treffens war der Punkt 3 praktisch umgesetzt und Punkt 4 konzeptioniert. In der Diskussion wurden Varianten der Umbenennung von Attributen beim „flach machen“ des Objektmodells und die Auswirkungen die Änderungen an dem bestehenden Modell haben können besprochen. Es wurde ein Konsens darüber erzielt, dass zunächst Schritt 4 nach dem Konzept des generischen Ansatzes von GDI-Service umgesetzt wird und das fertige „flache“ und vollständige OGR Modell auf der Web-Site des PostNAS Projektes noch mal zur Diskussion gestellt wird. Des Weiteren wurde vereinbart, dass das Endresultat des Projektes schließlich auf dem nächsten Anwendertreffen in Münster im Dezember präsentiert wird.

8 Fehlerprotokoll

Fehlernr. 11:

Das Attribut ogc_fid war in allen Tabellen des vorhandenen Datenbankmodells enthalten. Um sicher zu stellen, dass dieses Attribut auch immer mit eindeutigen Werten pro Tabelle gefüllt werden wurde es auf Typ serial gesetzt. Das Attribut ogc_fid wird jedoch nicht im AAA-Modell geführt. Es ist nur im Datenbankschema enthalten, weil es vorher schon drin war und das Modell so wenig wie möglich geändert werden sollte.

ToDo: Pfüren was da nach ogr2ogr Import drin steht und ob 4byte integer gerechtfertigt ist. Ggf. int 8 draus machen. Wenn ogr2ogr die Werte aus dem GML nimmt, auf den Typ setzen, der in gml verwendet wird und nicht mehr auf serial.

Fehlernr. 12:

Auch das Attribut identifier wurde aus dem vorhandenen Datenmodell übernommen. Dort hatte es den Typ character varying. Hier trägt ogr2ogr die gml_id ein. Man könnte den Typ also mit einer festen Länge, die einer zulässigen gml_id, versehen.

ToDo: Wie lang werden gml_ids in NAS-Dokumenten und deren Länge als Begrenzung verwenden.

Was ist mit dem Feld identifikator aus aa_objekt?

Fehlernr. 13:

a)

advstandardmodell ist vom Typ aa_advstandardmodell, was im UML Modell eine Enumeration ist. Der Typ der darin vorkommenden Werte ist nicht festgelegt. Er wird mit der php-Funktion ctype_digit ermittelt. Für den ersten Wert aus aa_advstandardmodell wird character ermittelt, daher wird der Typ in den Postgres Type character varying umgesetzt.

sonstigesmodell ist vom Typ aa_weiteremodellart, was im UML-Modell eine CodeListe ist. Die Werte der CodeListe sind nicht näher bestimmt, daher wurde hier der größt mögliche Typ für Werte verwendet und das ist text.

b)

Der Typ hängt also davon ab ob es eine CodeListe ist, dann immer text, oder eine Enumeration, dann in Anhängigkeit vom ersten Wert integer oder character varying.

CodeListen werden in der Regel extern definiert, so dass man nicht weiß welche Werte darin vorkommen könnten.

Der Typ AA_weitereModellart hat im Gegensatz zu anderen allerdings auch Werte im UML-Diagramm. Das heißt man könnte von diesen Werten auch einen character varying oder integer ableiten. Ebenso könne man annehmen, dass alle CodeListen nur Werte bis maximal 255 Zeichen aufnehmen können und bei CodeListen immer character varying nehmen.

ToDo: Prüfen ob CodeListen Values haben und diese ggf. in einer Tabelle abbilden wie eine Enumeration.

Fehlernr. 14:

Das hatten wir schon mal anders. Vorher war es so, dass die Kardinalität nur vom letzen Attribut entnommen wird.

AA_Objekt.lebenszeitintervall.AA_Lebenszeitinterval.endet war NULL, weil AA_Lebenszeitinterval.endet die Kardinaltiät 0 hatte. Das war nicht korrekt, weil AA_Objekt.lebenszeitintervall ja die Kardinalität 1 hatte.

In der letzten Versionsänderung haben wir dann also alle Attribute auf NOT NULL gesetzt wenn nur irgend ein Attribut im Pfad Kardinalität 1 hatte.

Das ist aber auch nicht Korrekt, weil natürlich endet NULL sein können muss. Nur die letzte Kardinaltät zu berücksichtigen geht aber auch nicht, weil es Fälle gibt, wo ein höheres Atttibut die Kardinalität 0..* haben kann und das letzte Attribut 1. z.B. postleitzahl in

AX_GeoreferenzierteGebaeudeadresse.postalischeAdresse.AX_Post.postleitzahl

postalische Addresse hat 0..1 aber postleitzahl 1. Das soll heißen wenn man eine postalische Adresse angibt, muss man auch eine Postleitzahl angeben.

Die richtige Umsetzung müßte also sein:

„Nur wenn alle Attribute im Pfad die Kardinaltität 1 haben, darf das Blattelement auf NOT NULL gesetzt werden.“ z.B. AA_Objekt.lebenszeitintervall.AA_Lebenszeitinterval.beginnt

Durch diese Regelung wird die Kardinalität viele Attribute aber nicht abgesichert. Das lässt sich am Beispiel AX_GeoreferenzierteGebaeudeadresse.postalischeAdresse.AX_Post.postleitzahl zeigen.

Obwohl laut UML-Modell die postleitzahl angegeben werden muss, wenn man eine postalische Adresse angibt, kann das Feld in der Datenbank leer bleiben.

Fehlernr. 15

zeigtAufExternes.AA_Fachdatenverbindung.art heisst mal zeigtaufexternes_art und mal einfach nur art, weil es mal doppelt vorkommt und umbenannt wurde und mal nicht. Insbesondere dann, wenn mehrere Attribute einer Klasse vom Typ AA_Fachdatenverbindung sind.

ax_regierungsbezirk hat z.B. das Attribut art.

So würde ogr2ogr das auch belegen, deshalb wurde das so umgesetzt.

Literaturverzeichnis

GeoInfoDok.....Adv, Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens (GeoInfoDok) Hauptdokument Version 6.0.1 Stand: 01.07.2009