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

Erstellt von: Dr. Peter Korduan, GDI-Sevice
letzte Änderung am: 04.04.2017

Kontakt:
GDI-Service Rostock
Dr. Peter Korduan
Joachim-Jungius-Straße 9
18059 Rostock
Tel: 0381/40344444
E-Mail: peter.korduan@gdi-service.de

Änderungen:

Datum	Änderung
22.03.2017	Punkt 9 Fortführung und den Fall mit den Enumerations hinzugefügt.
03.04.2017	Beschreibung zur Ermittlung der Differenz zwischen altem und neuem Schema hinzugefügt.
	Ergänzung zur Installation von libxml in ruby
04.04.2017	Ergänzungen zum Überschreiben nach xsdEncodingRule=iso19139_2007
	Löschen des Abschnittes 10: Fortführungen

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 7 enthält eine Schritt für Schritt Anleitung zum Einlesen von NAS-Dateien in das neue vervollständigte ogr Schema.

Das letzte Kapitel 15 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 1). 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 in der Konfigurationsdatei conf/model_aaa_conf.php angegeben werden welche Paket eingelesen werden sollen. Das Feld „BasePackageauswahl/-eingabe“ wird genutzt um anzugeben ob man von einem bestimmten Basispaket aus einlesen möchte. Sollen alle angegebenen 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.

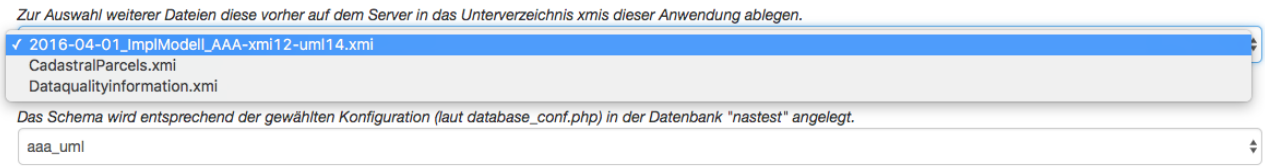


Abbildung 1: Auswahl vorhandener xmi Dateien

Die Option Argo Export mit ISO 19136 ist experimentell und wird nicht weiter gepflegt.

Nach betätigen des Button „Fülle DB mit XML-Inhalten“ wird ggf. das Schema angelegt und es startet das Einlesen der UML-Elemente in der Datenbank.

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.

Die in der Auswahllisten für das Schema zur Verfügung stehenden Schemanamen können in conf/model_aaa_conf.php vordefiniert werden. Wird nichts ausgewählt wird der Wert aus Konstante UML_SCHEMA aus conf/database_conf.php verwendet. Das model_aaa_conf.php verwendet werden soll wird im Parameter SCHEMA_CONF_FILE in conf/database_conf.php festgelegt.

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 Postgres-Datentyp und für jede Aufzählung einen Enumerations-Typ 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.

Diese Funktion ist für das Erstellen des OGR Schemas nicht notwendig und nur informativ oder für den Fall, dass nicht flache und objektorientierte Tabellen bevorzugt werden.

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. Damit ergibt sich das Flache Modell was der ogr2ogr Treiber für NAS-Dateien benötigt.

Zum Erzeugen des Schemas wählt man das UML-Schema und das Schema in dem das ogr Modell angelegt werden soll, siehe Abbildung 2. Die Auswahlmöglichkeiten werden in der Konfigurationsdatei conf/database_conf.php in den Konstanten UML_SCHEMA und OGR_SCHEMA gesetzt.

Nach dem Klick auf den Button „Erzeuge OGR-Schema“ wird das SQL-des Schemas erzeugt und ausgegeben. Diese SQL-Datei kann dann zum Anlegen des OGR-Schemas verwendet werden.

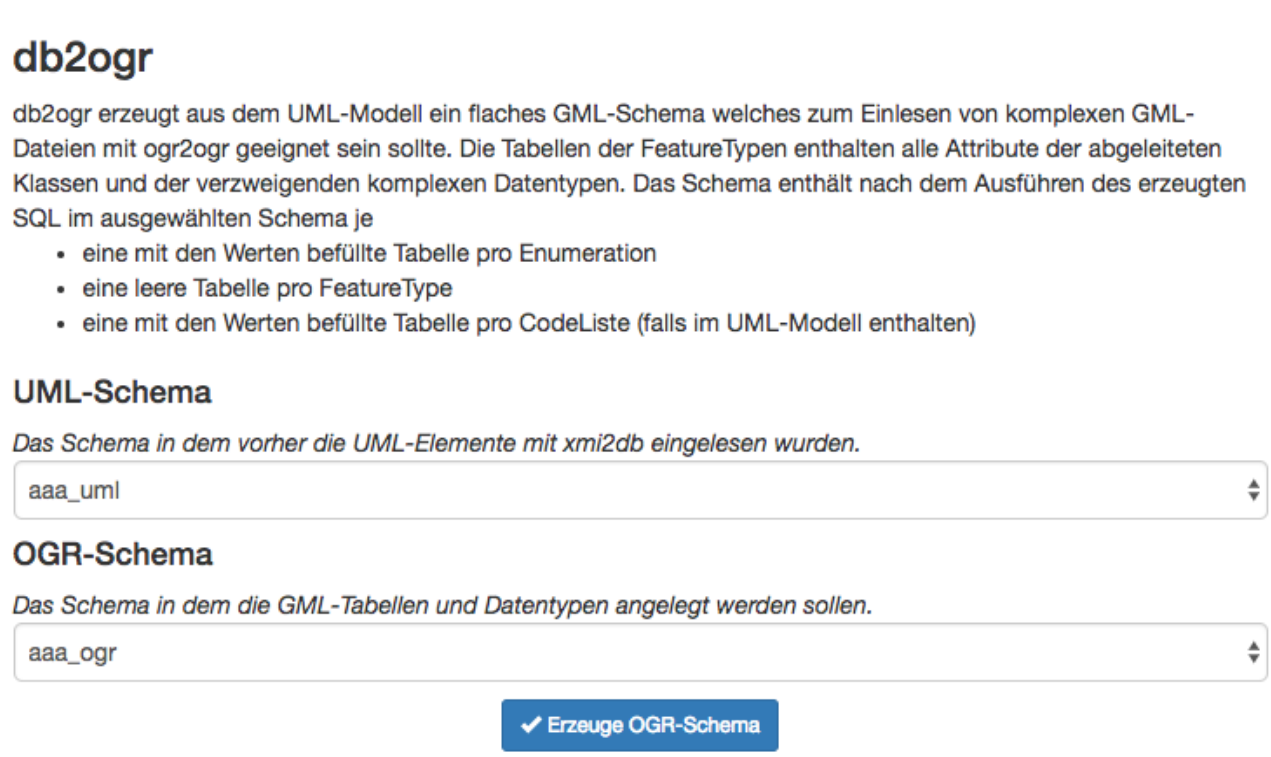


Abbildung 2: Startseite zum Erzeugen des ogr Modells
Die aktuelle Version des Schemas, die ausgewählten Pakete sowie der verwendete Filter werden am Anfang des Schemas als Kommentar ausgegeben, siehe Abbildung 1.

```
-- Version vom 22.11.2016 11:24
-- gewählte Pakete: 'Data quality information', 'AAA Basisschema', 'AAA_Basisklassen', 'AAA_GemeinsameGeometri
-- gewählte Filter: Ohne Attribute objektkoordinaten.
DROP SCHEMA IF EXISTS aaa_ogr CASCADE;
CREATE SCHEMA aaa_ogr;
COMMENT ON SCHEMA aaa_ogr IS 'Version vom 22.11.2016 11:24';
SET search_path = aaa_ogr, public;
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";


CREATE TABLE IF NOT EXISTS aa_advstandardmodell (
        wert character varying,
        beschreibung character varying,
        CONSTRAINT aa_advstandardmodell_pkey PRIMARY KEY (wert)
) WITH OIDS;

COMMENT ON TABLE aa_advstandardmodell IS 'Alias: "AA_AdVStandardModell", UML-Typ: Enumeration';
INSERT INTO aa_advstandardmodell (wert,beschreibung) VALUES
('DLKM', 'LiegenschaftskatasterModell'),
('DKKM500', 'KatasterkartenModell500'),
('DKKM1000', 'KatasterkartenModell1000'),
```

Abbildung 3: Kopf der Schemadatei

5.1 UML-Datentypen

Boolean

Im Projekt wurde per Konsens festgelegt, dass Typen, die in UML als Boolean definiert sind als character varying umgesetzt werden. Somit werden Warnungen vom der aktuelle OGR-Treiber unterdrückt.

Datetime

Datentypen, die in UML DateTime sind werden auch in der Datenbank als timestamp umgesetzt, auch wenn diese Felder in der Vorgängerversion des OGR-Schemas character varying waren.

Linestring

Da es NAS-Dateien gibt in denen Geometrie vom Typ LineString definiert sind, obwohl sie laut UMLModell

MultiLineString sein müssten wurden die UML-Typen LineString alle als Geometry in PostGIS umgesetzt.

Die Einstellung kann in conf/database_conf.php in der Konstante LINESTRING_AS_GEOMETRY mit true gesetzt oder fals aufgehoben werden.

5.2 Externe Datentypen

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

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 Specification mit einfachen Typen abgebildet.

Transaction
- content text

Query
- url character varying

Die FeatureTypen DQ_AbsoluteExternalPositionalAccuracy und DQ_RelativeExternalPositionalAccuracy sind beide von DQ_Element abgeleitet und haben selber keine Attribute. DQ_Element ist mit folgenden Attributen umgesetzt worden:
- nameOfMeasure character varying
- measureIdentification text
- measureDescription character varying
- evaluationMethodType character varying
- evaluationMethodDescription character varying
- evaluationProcedure text
- dateTime timestamp
- result text[]

Die Attribute measureIdentification (MD_Identifier), evaluationProcedure (CI_Citation) und result (DQ_Result) sind komplexe Datentypen, wurden aber als Text umgesetzt. Diese Lösung wurde als Konsens festgelegt um eine rekursive endlose Vertiefung des Modells zu verhindern. Anderenfalls gäbe es keine Möglichkeit das Modell in abgeflachter Form darzustellen.

5.3 Externe Klassenattribute

Im Modell werden Klassen als Datentypen verwendet, für die jedoch keine Attribute im Modell enthalten sind. Dazu wurden folgende Attribute entsprechend er externen Modelle in den Quellcode in schema.php aufgenommen:

Für die Klassen
- DQ_AbsoluteExternalPositionalAccuracy
- DQ_RelativeExternalPositionalAccuracy
- DQ_RelativeInternalPositionalAccuracy

wird die Klasse DQ_Element verwendet.

Klasse DQ_Element:
- nameOfMeasure: CharacterString [0..*]
- measureIdentification: MD_Identifier [0..1]
- measureDescription: CharacterString [0..1]
- evaluationMethodType: DQ_EvaluationMethodTypeCode [0..1]

- evaluationMethodDescription: CharacterString [0..1]
- evaluationProcedure: CI_Citation [0..1]
- dateTime: DateTime [0..*]
- result: DQ_Result [1..2]

Klasse LI_Lineage:

- statement: Character String [0..1]
- source: LI_Source [0..*] (Aggregation)
- processStep: LI_ProcessStep [0..*] (Aggregation)

Klasse LI_ProcessStep

- description: CharacterString
- rationale: CharacterString [0..1]
- dateTime: DateTime [0..1]
- processor: CI_ResponsibleParty [0..*]
- source: LI_Source [0..*] (Assoziation)

Klasse: LI_Source

- description: CharacterString [0..1]
- scaleDenominator: CharacterString [0..1]
- sourceReferenceSystem: CharacterString [0..1]
- sourceCitation: CI_Citation [0..1]
- sourceExtent: CharacterString [0..*]
- sourceStep: CharacterString [0..*] (Assoziation)

sourceStep ist eigentlich vom Typ LI_ProcessStep. Um aber Rekursionen zu vermeiden wurde sourceStep mit dem nicht komplexen Typ CharacterString besetzt, siehe auch Abbildung 4.

5.4 Überschreiben durch xsdEncodingRules

Im AAA Modell kommt es vor, dass externe Datentypen verwendet werden, deren Attribute aber überschrieben werden. Das trifft zu für das Attribut description aus den Klassen LI_ProcessStep und LI_Source. Zur gänzlichen Verwirrung werden diese Attribute auch noch kontextabhängig unterschiedlich verwendet.

LI_Lineage kommt vor in den komplexen Typen:

- AX_DQOhneDatenerhebung
- AX_DQMitDatenerhebung
- AX_DQPunktort

LI_Lineage hat LI_ProcessStep und LI_Source, die beide die Eigenschaft description vom Typ Character String haben. Dieser Typ gco:CharacterString kann durch folgende im ALKIS Modell vorhandene Typen überschrieben werden:

- AX_Datenerhebung_Punktort
- AX_LI_ProcessStep_Punktort_Description
- AX_LI_ProcessStep_MitDatenerhebung_Description
- AX_LI_ProcessStep_OhneDatenerhebung_Description
- AX_Datenerhebung
- AX_DQErfassungsmethode
- AX_DQErfassungsmethodeBesondererHoehenpunkt
- AX_DQErfassungsmethodeGewaesserbegrenzung
- AX_DQErfassungsmethodeMarkanterGelaendepunkt
- AX_DQErfassungsmethodeSekundaeresDGM
- AX_DQErfassungsmethodeStrukturierteGelaendepunkte

Diese sind im UML-Modell im taggedValue 'xsdEncodingRule' mit dem Wert iso19139_2007 gesetzt.

Die Struktur von LI_Lineage ist wie folgt:

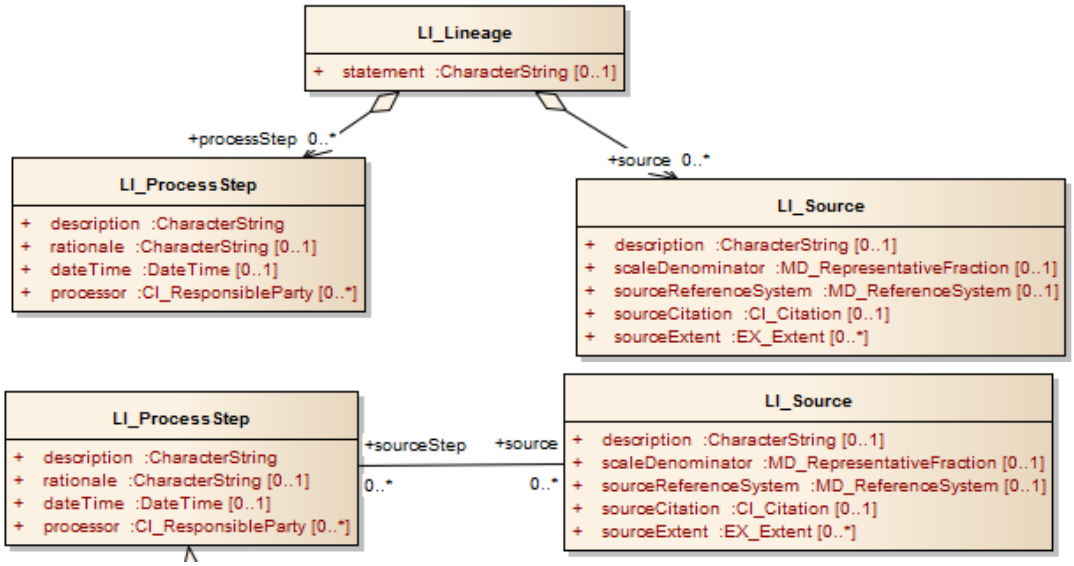


Abbildung 4: UML-Modell von LI_Lineage, LI_Source und LI_ProcessStep

Es gibt also 3 Stellen mit description vom Typ CharacterString, die überschrieben werden.

- LI_Lineage | source | description
- LI_Lineage | processStep | description
- LI_Lineage | processStep | source | description

Allerdings gibt es description nur einmal im Typ LI_Source und einmal im Type LI_ProcessStep.

Die GeoInfoDoc schreibt zur Verwendung der Typen:

„Soll Erhebung und/oder Erhebungsstelle dokumentiert werden, dann erfolgt dies über LI_ProcessStep- und LI_Source-Elemente. Die Erhebungsstelle wird in einem LI_ProcessStep mit self.description = 'Erhebung' und der Erhebungsstelle in self.processor dokumentiert. Die Datenerhebung wird in einem LI_Source-Element dokumentiert (über die Kennung aus der CodeList AX_Datenerhebung).“

Es ist nicht herauszubekommen ob bzw. wann source description und wann processStep source description verwendet wird.

- AX_DQMitDatenerhebung nutzt z.B. in NAS source description
- AX_DQPunktort hingegen processStep source description

In der folgenden Tabelle ist aufgeführt in welchem Kontext das Attribut description welchen ALKIS-Typ besitzt. Dabei ist nur source description aufgeführt welches aber gleichermaßen für processStep source description gilt.

Elternklasse	description in Klasse	Enumerationstyp
AX_DQOhneDatenerhebung	processStep	AX_LI_ProcessStep_OhneDatenerhebung_Description
	source	AX_Datenerhebung
AX_DQMitDatenerhebung	processStep	AX_LI_ProcessStep_MitDatenerhebung_Description
	source	AX_Datenerhebung
AX_DQPunktort	processStep	AX_LI_ProcessStep_Punktort_Description
	source	AX_Datenerhebung_Punktort

Tabelle 5.1: Belegung von LI_Lineage definition durch AAA-Enumerationstypen

Die Enumerations, die mit „AX_DQErfassungsmethode“ beginnen, überschreiben laut AAA-Fachschema.xsd zwar auch gco:CharacterString. Es handelt sich aber nicht um description vom Typ LI_Lineage oder deren Aggregationen LI_Source oder LI_ProcessStep.

Allerdings werden die Enumerations auch explizit in den NAS-Dateien als Tags aufgeführt, genauso wie

beim Überschreiben der description Typen in LI_Lineage.

Auf diese Art und Weise hat z.B. das Attribut description noch den Tag des Enumerationstyp AX_DQErfassungsmethode, während z.B. das Attribut identifikation, welches auch vom Stereotype Enumeration ist (AX_Identifikation) keinen Tag hat.

```
<AX_Erfassung_DGM>
        <description><AX_DQErfassungsmethode>5000</AX_DQErfassungsmethode></description>
        <identifikation>5400</identifikation>
</AX_Erfassung_DGM>
```

In diesem Beispiel kann man sehen, dass ganz allgemein gesagt in description Tags die Enumeration-Typen als Tags auftauchen, wenn sie in xsd als Überschreibungen für gco:CharacterString definiert sind.

Im OGR-Modell wird immer der Name des Blattelementes für die Attribute der flachen Tabellen verwendet. Daher muss das Attribut nach xsdEncodingRule durch den Namen der Enumerations-Typen überschrieben werden. Dies geschieht im Programm xmi2db im Modell Attribute in der Funktion overwriteIso19139Type.

Danach werden Attributnamen durch den Namen des Enumerations-Typ überschrieben, wenn er mit xsdEncodingRule=iso19139_2007 getagged ist.

Des Weiteren werden Attribute nach folgendem Schema entsprechend Tabelle 5.1 überschrieben.

5.5 Umbenennungen

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
```

Neben den Umbenennungen die notwendig waren um Doppelungen zu vermeiden wurden alle Elemente mit dem Namen zeigtAufExternes in zeigtaufexternes_ umbenannt. Da zeigtAufExternes kein Blattelement ist, spiegelt sich das nicht in dem Schema wieder. Diese Umbenennung wurde lediglich vorgenommen damit auch diese Elemente eingelesen werden, bzw. deren Blätter. Sobald ogr Treiber wieder zeigtAufExternes einliest kann diese Umbenennung entfallen. Die Einstellung zur Umbenennung kann in der Konstante RENAME_ZEIGT_AUF_EXTERNES in conf/database_conf.php mit true oder false angepasst werden.

5.6 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. Zur Bezeichnung des Filters steht die Konstante FILTER_INFO in conf/database_conf.php zur Verfügung. Diese Info wird im Kopf des Datebankschemas im Komentar ausgegeben, siehe Abbildung 1.

Länderfilter

Im Ordner conof/samples wurden für jedes der beteiligten Bundesländer Beispieldateien angelegt. Das Bundesland MV hat einen Filter geliefert. Die anderen für sl und rlp sind noch leer. Zur Benutzung dieser Filter werden die Beispieldateien nach conf/ copiert und dort angepasst. Der Name der verwendeten Filterdate kann in der Konstante FILTER_FILE in conf/database.conf eingestellt werden.

6 Differenz zwischen altem und neuem Schema

Zum Vergleich wurden folgende Schemata herangezogen

6.1 Altes PostNAS-Schema

Das Alte PostNAS-Schema wurde am 03.04.2017 mit der Datei alkis_PostNAS_schema.sql aus dem svn Repository: http://trac.wheregroup.com/PostNAS Version @ 378 in einer neuen Datenbank postnas_org angelegt.

Dazu wurde das Repository zunächst heruntergeladen mit:

```
svn co https://svn.wheregroup.com/PostNAS postnas_org --username 'nas-guest'
--password 'guest'
```

Im Verzeichnis trunk/import wurde zunächst ein Link alkis-trigger.sql auf die Datei alkis-trigger-kill.sql angelegt, dann eine leere Datenbank postnas_org mit PostGIS Erweiterung angelegt

```
CREATE DATABASE postnas_org;
CREATE EXTENSION postgis;
```

und anschließend die Datei alkis_PostNAS_schema.sql mit psql ausgeführt.

```
psql -h pgsql -U kvwmap -f alkis_PostNAS_schema.sql -v alkis_epsg="25832"
postnas_org
```

Es werden mit dem Skript 136 Tabellen im Schema public angelegt, siehe Abbildung 5.

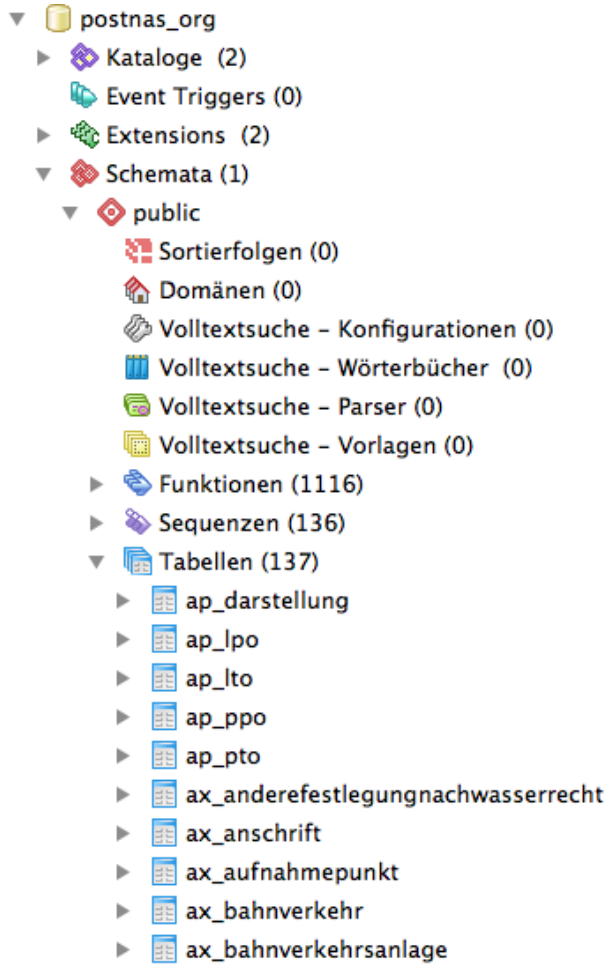


Abbildung 5: altes PostNAS Schema

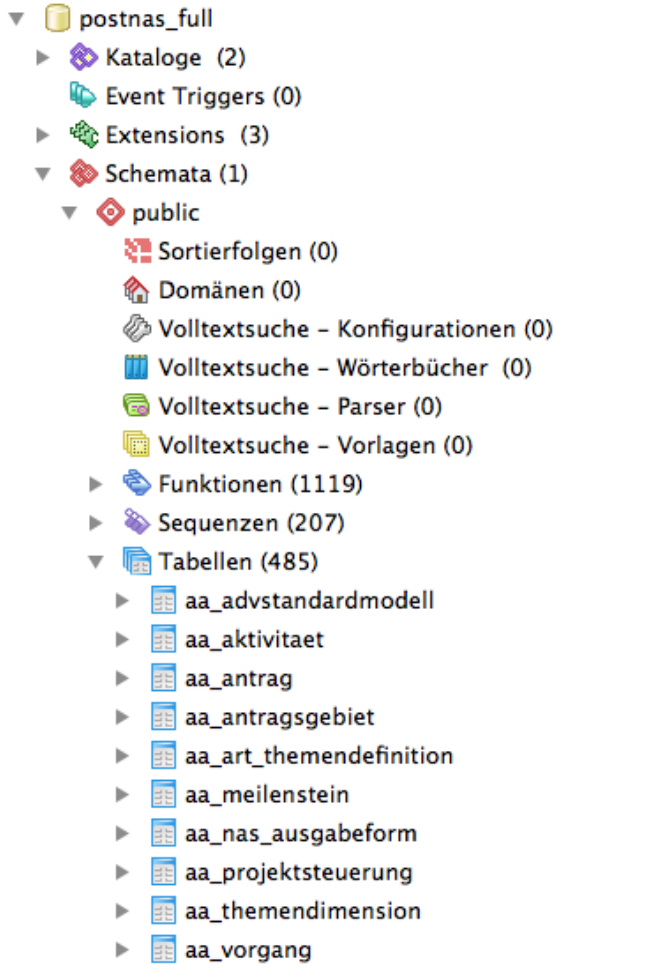


Abbildung 6: neues PostNAS Schema

6.2 Neues PostNAS-Schema

Das neue vollständige PostNAS-Schema wurde am 03.04.2017 mit xmi2db in der Version vom 22.03.2017 17:39 und der folgenden URL generiert.

```
http://gdi-service.de/xmi2db/converter/db2ogr.php?
umlSchema=aaa_uml&ogrSchema=aaa_ogr&epsgCode=25832&withCodeLists=0
```

Zum Anlegen des Schemas wurde die Datenbank postnas_full angelegt und die PostGIS-Erweiterung eingespielt:

```
CREATE DATABASE postnas_org;
CREATE EXTENSION postgis;
```

Zur besseren Vergleichbarkeit wurde das neue Schema für den Vergleich auch in einem public Schema angelegt. Dazu wurden die ersten Zeilen im Script vom xmi2db wie folgt angepasst

```
--DROP SCHEMA IF EXISTS aaa_ogr CASCADE;
--CREATE SCHEMA aaa_ogr;
COMMENT ON SCHEMA public IS 'Version vom 22.03.2017 17:39';
SET search_path = public;
```

und der Text aaa_ogr im Funktionenteil des Skriptes durch public ersetzt.

Durch Ausführen des Skriptes in einem SQL-Client entstehen im Schema 484 Tabellen, siehe Abbildung 6.

Die Anzahl + 1 entsteht, weil die Tabelle spatial_ref_sys schon durch PostGIS vorher existiert. Beide Schemata haben das exakt gleiche PostGIS.

6.3 Differenzbildung

Für die Differenzbildung wird das Werkzeug apgdiff eingesetzt, siehe https://apgdiff.com.

Dazu sind zunächst von beiden Datenbanken Dumps anzulegen.

```
pg_dump -h pgsql -U kvwmap -s -O -x -Fp -f postnas_org_dump.sql postnas_org
pg_dump -h pgsql -U kvwmap -s -O -x -Fp -f postnas_full_dump.sql postnas_full
```

Anschließend werden die Dateien auf die Seite https://apgdiff.com/diff_online.php hochgeladen und die Differenzdokumentation erstellt.

Das Ergebnis ist eine SQL-Datei, mit der vom Schema postnas_org in die Struktur vom Schema postnas_full gewechselt werden kann. Die Datei steht auch zum Download unter der folgenden Adresse zur Verfügung:

http://gdi-service.de/xmi2db/sql/postnas_diff.sql

7 Schritt für Schritt Anleitung

7.1 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/

7.2 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 7 erfordert die Installation von libxml-ruby. Dieses Debianpaket wird wie folgt installiert:

```
apt-get update && apt-get install \
  ruby1.9.3 \
  ruby1.9.3-dev \
  libxml-ruby
gem install libxml-ruby
Bei Verwendung von rvm die Umgebung setzen.
source /etc/profile.d/rvm.sh
```

Falls die Installation von libxml-ruby nicht funktioniert liegt das vielleicht an einer veralteten Version. Libxml-ruby sollte mindestens die Version 2.9.0 haben. Das kann abgefragt werden mit:
puts XML::VERSION
Unter Debian 8 heißt die Bibliothek ruby-libxml.

7.3 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.

7.3.1 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.

7.3.2 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 7.1.

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

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

7.4 Einlesen

7.4.1 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

7.4.2 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
```

8 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.

9 Fragen-, Antwortprotokoll

9.1 Frage 1

Frage:

Attribute mit fehlender Multiplizität

Beim Blick in aaa_ogr_2016-09-01.sql :

```
CREATE TABLE ax_flurstueck (
   gml_id text NOT NULL,
   ....
   angabenzumabschnittbemerkung text,
   angabenzumabschnittflurstueck text,
   angabenzumabschnittnummeraktenzeichen text,
   angabenzumabschnittstelle text,
   flaechedesabschnitts double precision,
   kennungschluessel text,
   zeitpunktderentstehung date,
   zustaendigestelle_land text,
  ...);
```

fällt bzgl. aller Bestandteile des Datentyps AX_SonstigeEigenschaften_Flurstueck (kennungschluessel, flaecheDesAbschnitts und ax_flurstueck.angabenzumabschnitt*) auf, dass hier jeweils kein Array (mit den []) vorhanden ist. Alle diese gehören zum Attribut sonstigeEigenschaften, welches 0..* mal auftauchen darf. Wie erklärt sich, dass eine multiple Eigenschaft ohne Array abgebildet wird?

Antwort:
Die Attribute hatten die Multiplizität des jeweiligen Blatt-Elementes des Modells. Jetzt werden alle Multiplizitäten entlang des Pfades verwendet. Wenn eines davon * hat oder > 1 wird der Typ auf [] gesetzt.

Es wurde die Kardinalität des letzten Attribut im Pfad berwendet. Es müssen aber auch die Kardinalitäten der übergeordneten Attribute berücksichtigt werden. Falls dort also ein * vorkommt müssen die letzten Attribute auch dem obersten Attribut mit [] zugewiesen werden.

9.2 Frage 2

Frage:

Wie kann eine multiple Befüllung von AX_Gebaeude.nutzung funktionieren?

In aaa_ogr_2016-09-01.sql ist folgendes zu AX_Gebaeude enthalten:

```
CREATE TABLE ax_gebaeude (

gml_id text NOT NULL, anlass aa_anlassart[], beginnt date, endet date, advstandardmodell aa_advstandardmodell, sonstigesmodell aa_weiteremodellart,

art character varying, fachdatenobjekt_name text, uri character varying, "position" public.geometry, anzahlderoberirdischengeschosse integer, anzahlderunterirdischengeschosse integer, baujahr integer[], bauweise ax_bauweise_gebaeude, dachart text, dachform ax_dachform, dachgeschossausbau ax_dachgeschossausbau_gebaeude, gebaeudefunktion ax_gebaeudefunktion, gebaeudekennzeichen text, geschossflaeche double precision, grundflaeche double precision, hochhaus boolean, lagezurerdoberflaeche ax_lagezurerdoberflaeche_gebaeude, name text[], anteil integer, nutzung ax_nutzung, objekthoehe double precision, herkunft text, umbauterraum double precision, weiteregebaeudefunktion ax_weitere_gebaeudefunktion[], zustand ax_zustand_gebaeude, istabgeleitetaus text[], traegtbeizu text[], hatdirektunten text[], inverszu_hatdirektunten text[], istteilvon text[], inverszu_dientzurdarstellungvon_ap_lto text[], inverszu_dientzurdarstellungvon_ap_pto text[], inverszu_dientzurdarstellungvon_ap_ppo text[], inverszu_dientzurdarstellungvon_ap_lpo text[], inverszu_dientzurdarstellungvon_ap_fpo text[], inverszu_dientzurdarstellungvon_ap_darstellung text[], inverszu_dientzurdarstellungvon_ap_kpo_3d text[], inverszu_gehoertzu text[], gehoert text[], zeigtauf text[], hat text, haengtzusammenmit text, gehoertzu text, inverszu_zeigtauf text[]
);
```

Nutzung wird hierbei mit ax_nutzung aufgeführt.

ax_nutzung führt in aaa_ogr_2016-09-01.sql zu:

```
CREATE TABLE ax_nutzung (

wert character varying NOT NULL,

beschreibung character varying );
```

Antwort:
ax_nutzung ist noch als Typ definiert und wird nicht zerlegt in zwei Attribute. ax_nutzung ist aber eine Enumeration und wird in der nächsten Version auch als solche definiert.

Wie gesagt ich arbeite gerade an beiden Problemen und sende Ihnen heute noch das korrigierte Schema zu.

Enumeration Typen werden jetzt nicht mehr als Datenbanktypen definiert, sondern als character varying, wenn die Werte der Aufzählung alphanumerisch sind und integer wenn es Zahlen sind.

CodeList Typen werden als Text umgesetzt, da hier Referenzen von Codelisten Registries reinkommen sollten.

Alle Enumerations und CodeListen haben eine eigene Tabelle. Enums mit Wert und Beschreibung, CodeListen mit codeSpace und id. Diese Tabellen sind befüllt wenn sie im UML-Modell vorhanden waren (Implementierungsmodell).

nutzung ax_nutzung ist jetzt also

nutzung character varying []

ToDo:

Das muss ich aber noch mal testen wie Codelisten Werte in NAS aussehen und mit ogr2ogr eingelesen werden. Kann sein, dass die Attribute noch aufgeteilt werden müssen in codeSpace und Id.

Lösung:

CodeListen werden nur mit Ihren Werten übergeben. z.B. anlass hat den Typ AA_Anlassart, übergeben wird nur der Wert, z.b. 080200

9.3 Frage 3

Frage:

Alle OA betroffen, hier am Beispiel AP_Darstellung:

CREATE TABLE aa_advstandardmodell (

wert character varying NOT NULL,

beschreibung character varying);

CREATE TABLE aa_weiteremodellart (id integer NOT NULL, name character varying, status character varying,

definition text,

additional_information text);

CREATE TABLE ap_darstellung (gml_id text NOT NULL, anlass aa_anlassart[], beginnt date,

endet date, advstandardmodell aa_advstandardmodell, sonstigesmodell aa_weiteremodellart, zeigtaufexternes_art character varying, name text, uri character varying, art character varying, darstellungsprioritaet integer, positionierungsregel text, signaturnummer character varying, istteilvon text[], inverszu_dientzurdarstellungvon_ap_lto text[], inverszu_dientzurdarstellungvon_ap_pto text[], inverszu_dientzurdarstellungvon_ap_ppo text[], inverszu_dientzurdarstellungvon_ap_lpo text[], inverszu_dientzurdarstellungvon_ap_fpo text[], inverszu_dientzurdarstellungvon_ap_darstellung text[], inverszu_dientzurdarstellungvon_ap_kpo_3d text[], dientzurdarstellungvon text[]

);

Wie kann damit ein Objekt mit mehreren Modellarten abgelegt werden?

Antwort:

Erledigt sich durch Korrektur der Multiplizität, siehe Fehler: Referenz nicht gefunden

9.4 Frage 4

Frage:
Alle OA betroffen, hier am Beispiel AP_Darstellung:
Wozu dienen inverszu_dientzurdarstellungvon_ap_lto text[],
inverszu_dientzurdarstellungvon_ap_pto text[], inverszu_dientzurdarstellungvon_ap_ppo text[],
inverszu_dientzurdarstellungvon_ap_lpo text[], inverszu_dientzurdarstellungvon_ap_fpo text[],
inverszu_dientzurdarstellungvon_ap_darstellung text[], inverszu_dientzurdarstellungvon_ap_kpo_3d text[],
?

Antwort:

die Attribute wie

inverszu_dientzurdarstellungvon_ap_lto text[]

sind die Relationen zu anderen Tabellen.

9.5 Frage 5

Frage:

Wie kann ein Objekt mit mehreren Fachdatenverbindungen (Set) abgelegt werden?

Antwort:

zeigtAufExternes Attribute sind jetzt auch []

Die jeweiligen Blattattribute sind daher auch [], man beachte aber, dass diese manchmal umbenannt wurden, wegen der sonst vorhandenen Doppelungen.

9.6 Frage 6

Frage:

Alle OA betroffen:

Wie ist bei beginnt bzw. endet „date“ definiert? Enthält es auch die Uhrzeit?

Antwort:

beginnt und endet war vorher in der Tat nicht genau genug.

Jetzt sind alle UML-Attributtypen datetime in postgres

timestamp without time zone

date bleibt date in postgres.

9.7 Frage 7

Frage:

AP_* betroffen:

istteilvon wird nicht benötigt.

Antwort:

sieht Frage 8

9.8 Frage 8

Frage:

AP_FPO, AP_PPO, AP_PTO, AP_LPO, AP_LTO, hier am Beispiel AP_FPO:

CREATE TABLE ap_fpo (

gml_id text NOT NULL, anlass aa_anlassart[], beginnt date, endet date, advstandardmodell aa_advstandardmodell, sonstigesmodell aa_weiteremodellart, zeigtaufexternes_art character varying, name text, uri character varying, "position" public.geometry, art character varying, darstellungsprioritaet integer, signaturnummer character varying, istabgeleitetaus text[], traegtbeizu text[], hatdirektunten text[], inverszu_hatdirektunten text[], istteilvon text[], inverszu_dientzurdarstellungvon_ap_lto text[], inverszu_dientzurdarstellungvon_ap_pto text[], inverszu_dientzurdarstellungvon_ap_ppo text[], inverszu_dientzurdarstellungvon_ap_lpo text[], inverszu_dientzurdarstellungvon_ap_fpo text[], inverszu_dientzurdarstellungvon_ap_darstellung text[], inverszu_dientzurdarstellungvon_ap_kpo_3d text[], dientzurdarstellungvon

);

text[] istabgeleitetaus text[],

traegtbeizu text[],

hatdirektunten text[],

inverszu_hatdirektunten text[],

werden nicht benötigt.

Antwort:

istteilvon ist aber im UML-Modell vorhanden. Wir haben erstmal alles übernommen.

Dinge, die nicht benötigt werden können in dem noch ausstehenden Filter herausgenommen werden. Gilt auch für alle anderen Dinge, die nicht benötigt werden, Beziehungen, Attribute oder ganze Tabellen.

9.9 Frage 9

Frage:

"position" public.geometry, Kann dies Flächen aus einem oder auch mehreren (ggf. getrennt liegenden)

Teilflächen abbilden?

Antwort:

Der Postgres Type geometry kann alles abbilden von Point über Multipoint, Line, Multiline, Polygon bis zu Multipolygon.

Getrennt liegende Flächen sind Multipolygone, also können diese abgebildet werden.

9.10 Frage 10

Frage:
inverszu_* sollen implementierungsseitig (nicht seitens anwenderspezifischer Filterung) entfallen. Da in AAA-konformen NAS-Daten keine Gegenrelationen enthalten sind, ist es sinnlos und ungünstig, dafür in PostGreSQL Tabellenstrukturen vorzuhalten – diese werden seitens NAS-Daten nicht befüllt. Falls es doch einen Grund für inverszu_* geben sollte, so möge dieser bitte benannt werden. Im Modell nicht benannte Gegenrelationen wie z.B. AX_Person inverszuIst AX_Benutzer werden dadurch auch vermieden.

Antwort:

Relationen, die mit inversZu beginnen werden weggelassen.

9.11 Frage 11

Frage:
ogc_fid wird bei allen Fachobjekten mit serial NOT NULL gesetzt.
Vgl. https://www.postgresql.org/docs/9.2/static/datatype.html: serial = autoincrementing four-byte integer.
Wozu dient dieser 4byte Integer?

Antwort:

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:

Prüfen 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.

Lösung:

Es wird serial verwendet. Die Werte sind vom Typ int4, der 2147483647 positive Werte pro Tabelle aufnehmen kann. Der gleiche Typ wurde auch im vorherigen ALKIS ogr Modell verwendet.

9.12 Frage 12

Frage:
Warum ist identifier vom Typ character varying und nicht fester Länge?

Antwort:

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?

Lösung:

gmi_ids sind immer 16 Stellen lang und werden jetzt auch als varchar (16) definiert.

9.13 Frage 13

Frage:

a) Warum hat advstandardmodell Datentyp character varying [] NOT NULL, sonstigesmodell hingegen text[] NOT NULL ?
b) Auch an anderen Stellen wird mal der eine, mal der andere Datentyp verwendet. Was sind die generellen Regeln für die jeweilige Verwendung?

Antwort:

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.

Lösung:

Die Frage erübrigt sich, da externe Codelisten auch extern geändert werden können. Daher können keine festen EnumTypen in dem Modell festgehalten werden.

9.14 Frage 14

Frage:
(Lebenszeitintervall)endet hat ein NOT NULL Constraint. Laut https://www.postgresql.org/docs/9.2/static/ddl-constraints.html und dort 5.3.2 Not-Null Constraints bedeutet dies, dass ein Wert gesetzt sein ***muss.*** Das wiederum bedeutete, dass alle Objekte in der Datenbank untergegangen wären, was verkehrt ist. Warum wird bei endet NOT NULL-Constraint gesetzt? Diese Frage bezieht sich auf alle optionalen Attribute (d.h. solche mit Kardinalitätsuntergrenze „0“), bei denen das Constraint NOT NULL gesetzt ist!
Konkrete Beispiele für letztgenannten Sachverhalt: zeigtaufexternes_art, name, uri bei AX_Benutzer ODER sonstigesmodell und bei AX_Regierungsbezirk ODER regierungsbezirk bei AX_LagebezeichnungKatalogeintrag ODER istamtsbezirkvon bei AX_Gemarkung ODER AX_Verband.regierungsbezirk sowie gemeindeteil ODER AX_BesondererBauwerkspunkt land sowie stelle. Fachlich gewichtig ist auch ax_georeferenziertegebaeudeadresse.postleitzahl text NOT NULL, da die VermKV selbst die PLZ nicht führt.

Antwort:

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.

9.15 Frage 15

Frage:
Zum Beispiel bei AX_Regierungsbezirk fehlt zeigtaufexternes_art. Warum? Es müsste bei allen Erben von AA_Objekt vorhanden sein.

Antwort:

Es ist richtig, dass alle von AA_Objekt erben und demzufolge die Attribute von zeigtaufexternes haben müssen. Das haben sie auch. Bei Objekten, wo es aber auch noch andere Attribute in den Blattelementen gibt die „art“ heißen, wird „art“ umbenannt in „zeigtaufexternes_art“
AX_Regierungsbezirk hat z.B. kein weiteres Attribut art, daher heißt hier das Attribut vom Objekt AA_Fachdatenverbindung, welches für die Beziehung zeigtaufexternes verwendet wird „art“.

AX_Reservierung hingegen hat selbst ein Attribut mit dem Namen „art“ vom Typ enumeration AX_Art_Reservierung. Deshalb wurde das Attribut art aus AA_Fachdatenverbindung in zeigtaufexternes_art umbenannt. Entsprechende Erläuterungen über die Herkunft findet sich im Kommentar der Attribute, siehe Abbildung 7.

```
CREATE TABLE aaa_ogr.ax_reservierung
(
  ogc_fid serial NOT NULL,
  identifier CHARACTER VARYING,
  gml_id TEXT NOT NULL,
  anlass TEXT[], -- anlass  codelist AA_Anlassart 0..*
  beginnt TIMESTAMP WITHOUT TIME ZONE NOT NULL, -- lebenszeitintervall AA_Lebenszeitintervall|beginnt  DateTime 1
  endet TIMESTAMP WITHOUT TIME ZONE, -- lebenszeitintervall AA_Lebenszeitintervall|endet  DateTime 0..1
  advstandardmodell CHARACTER VARYING[] NOT NULL, -- modellart AA_Modellart|advStandardModell enumeration AA_AdVStandardModell 1
  sonstigesmodell TEXT[] NOT NULL, -- modellart AA_Modellart|sonstigesModell codelist AA_WeitereModellart 1
  zeigtaufexternes_art CHARACTER VARYING[], -- zeigtAufExternes AA_Fachdatenverbindung|art  URI 1
  NAME TEXT[], -- zeigtAufExternes AA_Fachdatenverbindung|fachdatenobjekt|AA_Fachdatenobjekt|name   1
  uri CHARACTER VARYING[], -- zeigtAufExternes AA_Fachdatenverbindung|fachdatenobjekt|AA_Fachdatenobjekt|uri  URI 1
  ablaufderreservierung DATE, -- ablaufDerReservierung   Date 0..1
  antragsnummer TEXT, -- antragsnummer    0..1
  art INTEGER NOT NULL, -- art  enumeration AX_Art_Reservierung 1
  auftragsnummer TEXT, -- auftragsnummer    0..1
  bezirk TEXT, -- gebietskennung AX_Reservierungsauftrag Gebietskennung|buchungsblattbezirk|AX_Buchungsblattbezirk_Schluessel|bezirk   1
```

Abbildung 7: Umbenennung von Attibut art in zeigtaufexternes_art wenn schon vorhanden

Es wird immer das was später kommt umbenannt so wie es auch in ogr2ogr umgesetzt wird.

Der Grund warum die Umbenennung nicht einheitlich gemacht wurde ist der, dass vorgegeben wurde, dass so wenig wie möglich umbenannt werden soll, bzw. nur das was notwendig ist.

9.16 Frage 16

Frage:
AX_BesondererBauwerkspunkt und weitere (z.B. AX_Sicherungspunkt, AX_BesondererGebaeudepunkt) haben im DB-Schema die Relation bestehtaus. Warum?

Antwort:

Die Klasse AA_ZUSO hat die Assoziation bestehtAus zu AA_Objekt, siehe Abbildung 8. Also dürften alle, die von AA_ZUSO abgeleitet sind auch diese Beziehung haben. AX_Sicherungspunkt ist von AX_Netzpunkt abgeleitet und diese Klasse von AA_ZUSO, usw.

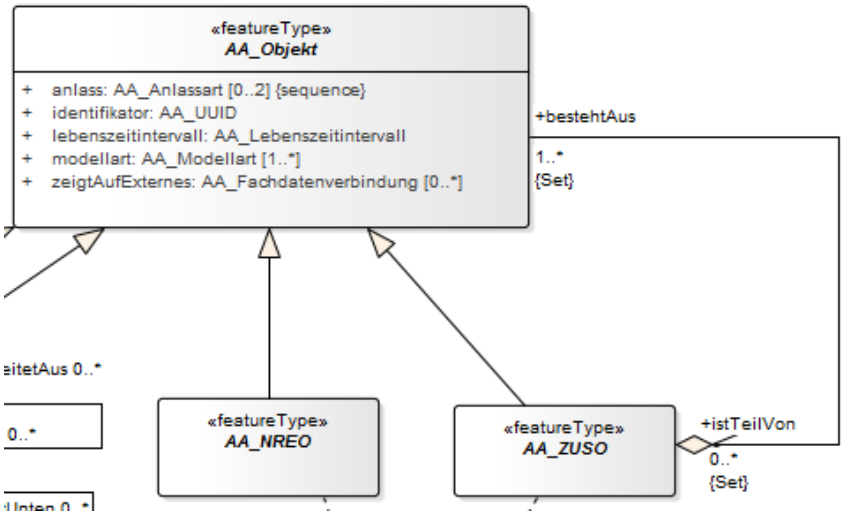


Abbildung 8: Beziehung zwischen AA_ZUSO und AA_Objekt

9.17 Frage 17

Frage:
Was ermöglicht geometry bei AP_FPO, AX_Polder, AX_HistorischesFlurstueck, AX_Kondominium, AX_Duene, AX_Transportanlage, AX_Gleis, AX_Bahnverkehrsanlage...?

Antwort:

Ich nehme an Sie meinen hier das Attribut position welches im ogr-Schema als geometry umgesetzt wurde.

AP_FPO ist abgeleitet von der Klasse AP_GPO aus dem Paket AAA_Praesentationsobjekt und von der Klasse AU_Flaechenobjekt aus dem Paket AAA_Unabhaengige Geometrie, siehe Abbildung 9.

AP_FPO ist also zur Präsentation von Objekten mit unabhängiger Geometrie geeignet.

AX_HistorischesFlurstueck ist von AU_Flaechenobjekt abgeleitet. Ich denke dort dient es auch zur Darstellung in einer Präsentation. Wozu nun genau welche Objekte eine Geometrie bekommen haben kann ich Ihnen nicht sagen. Das haben die Modellierer der AdV so gewollt. Wozu man es dann benutzen ist hier jedoch eindeutig. Zur Darstellung in Ausgaben. Bei einigen Objekten sicher auch zur Suche, Filterung, Verschneidung etc. Der Gründe gäbe es viele.

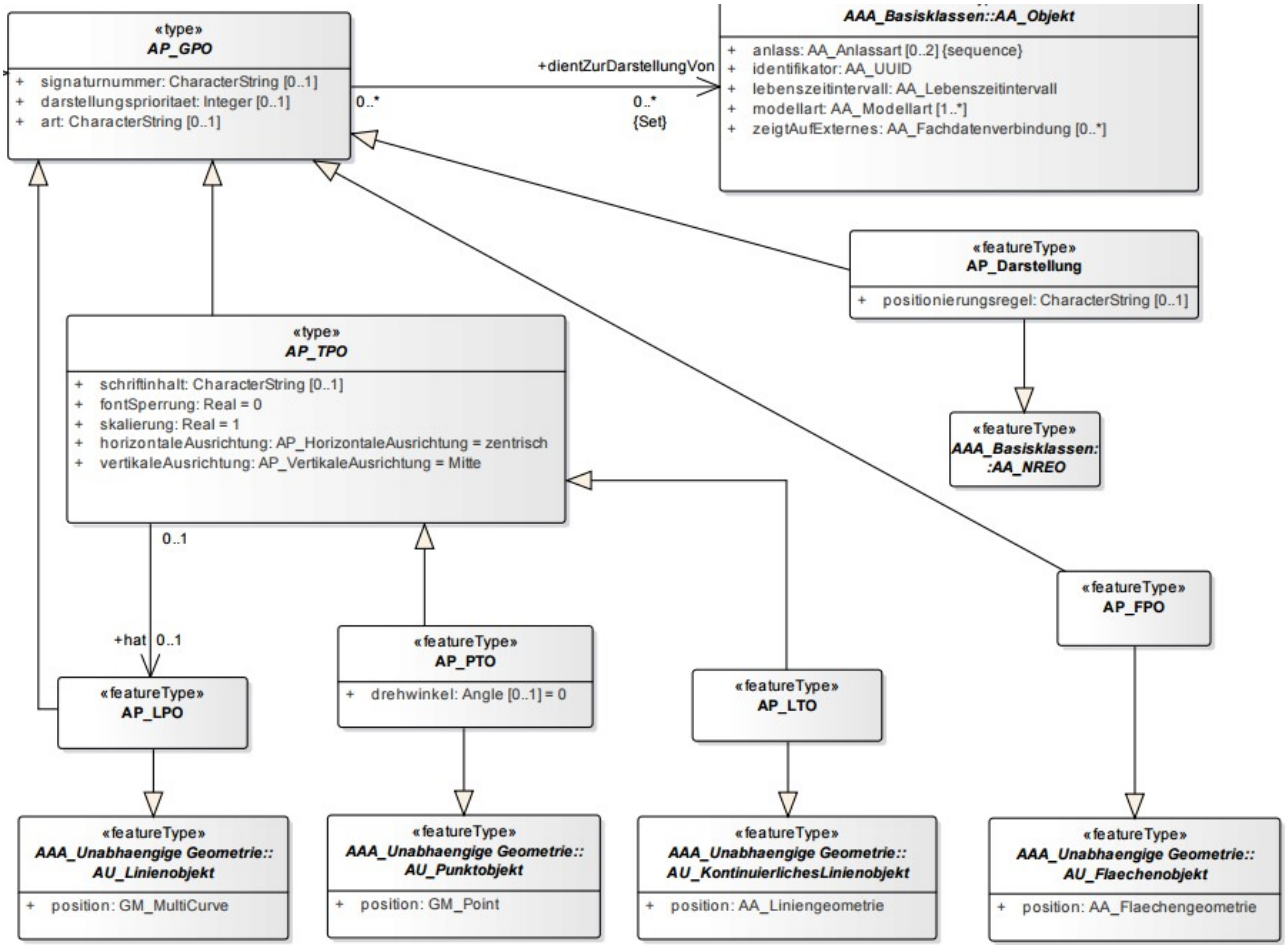


Abbildung 9: Zusammenhang AP_FPO mit AP_GPO und AU_Flaechenobjekt

9.18 Frage 18

Frage:
AX_Geripplinie.hoehengenauigkeit und alle Elemente, die DQ_AbsoluteExternalPositionalAccuracy nutzen: Abgebildet wurde auf text. Wird das für die ISO 19139-Konstrukte reichen?

Antwort:

Das ist ein sehr schwieriger Punkt, den ich aber schon angesprochen hatte.
Die Klasse DQ_AbsoluteExternalPositionalAccuracy wird sowohl im AAA-Modell als auch im AAA-Implementierungsmodell als Classifier genutzt, ist aber im AAA-Implementierungsmodell, welches wir umsetzen sollten, überhaupt nicht vorhanden, so wie alle externen Typen nicht vorhanden sind.
Um diese Typen von externen Modellen im aaa_ogr Modell nutzen zu können, müsste man also entweder das AAA-Implementierungsmodell um die UML-Dinge erweitern, die AAA-Klassen verwenden oder die Attributierung nachträglich durch vorbereitete Definitionen hinzufügen, z.B. bei der Datenbankschemaerstellung.
Das wird jedoch nicht ganz einfach, was man am Typ DQ_AbsoluteExternalPositionalAccuracy, der in ISO 19115 beschrieben ist (und die Umsetzung in ISO 19139), belegen kann. Wie man in Abbildung 10 sehen kann ist die Klasse von DQ_Element und DQ_PositionalAccuracy abgeleitet.
DQ_AbsoluteExternalPositionalAccuracy selbst und DQ_PositionalAccuracy haben keine Attribute, aber DQ_Element, siehe Abbildung 11. Der dort verwendete komplexe Typ MD_Identifier und die CodeListe DQ_EvaluationMethodTypeCode könnte man ja vielleicht noch einfach umsetzen aber wie sieht es mit DQ_Result aus. Das könnte DQ_ConformanceResult oder DQ_QuantitativeResult sein, wie Abbildung 12 belegt. Welche Klasse ist aber nicht genauer spezifiziert.

Ähnlich kompliziert wird es bei CI_Citation. Schauen Sie mal bitte in Abbildung 12. Dadurch würden, wenn man alle Attribute, auch der verwendeten SubTypen berücksichtigt mehr als 30 zusätzliche Attribute im Modell vorgehalten werden, siehe folgende Liste. Außerdem würde sich hier eine Rekursion mit

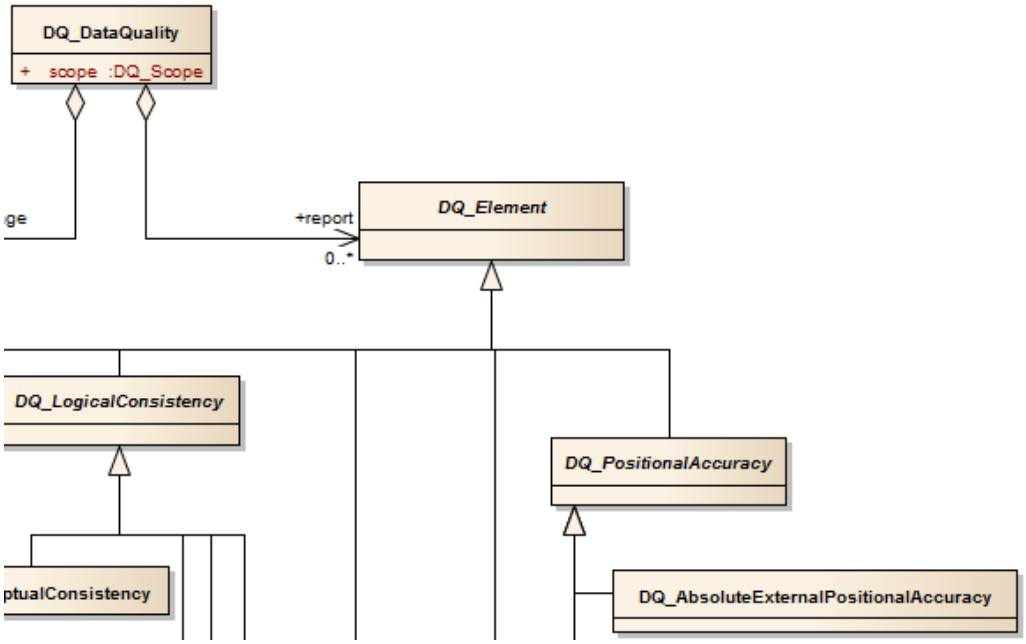


Abbildung 10: Struktur der Klassenhierarchie von DQ_AbsoluteExternalPositionalAccuracy
Endlosschleife ergeben, da das Attribut authority auch vom Typ CI_Citation ist.

<<datatype>> CI_Citation

- Title CharacterString
- alternateTitle CharacterString
- date CI_Date
  - date Date
  - dateType CI_DateTypeCode
- edition CharacterString
- editionDate Date
  - date Date
  - dateType CI_DateTypeCode
- identifier MD_Identifier
  - authority CI_Citation
    - noch mal alle Attribute von CI_Citation!!!
  - code CharacterString
- citeResponsibleParty CI_ResponsibleParty
  - individualName CharacterString
  - organisationName CharacterString
  - positionName CharacterString
  - contactInfo CI_Contact
    - phone
    - address
    - onlineResource CI_OnlineResource

- linkage URL
- protocol CharacterString
- applicationProfile CharacterString
- name CharacterString
- description CharacterString
- description CharacterString
- function CI_OnLineFunctionCode

  - hoursOfService CharacterString
  - contactInstruction CharacterString

- role CI_RoleCode

- presentationForm CI_PresentationFormCode
- series CI_Series
  - name CharacterString
  - issueIdentification CharacterString
  - page CharacterString
- otherCitationDetails CharacterString
- collectiveTitle CharacterString
- ISBN CharacterString
- ISSN CharacterString

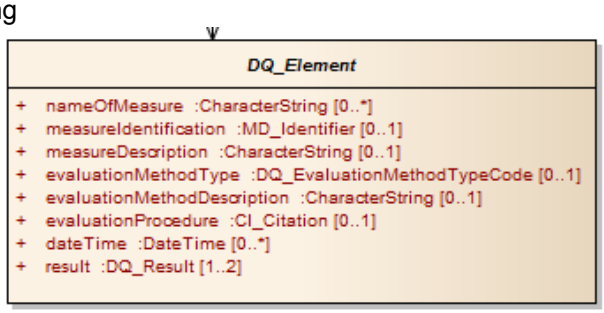


Abbildung 11: Eigenschaften von DQ_Element

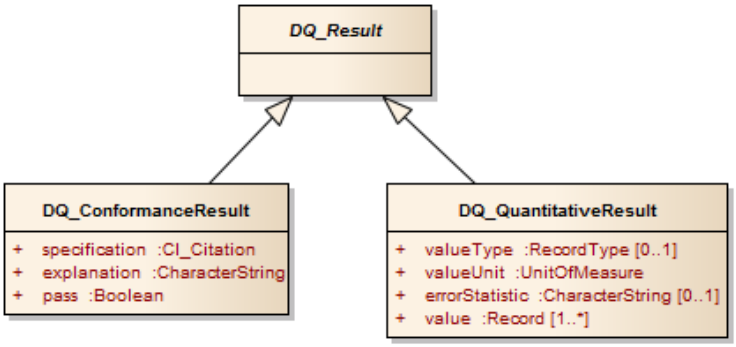


Abbildung 12: Varianten von DQ_Result

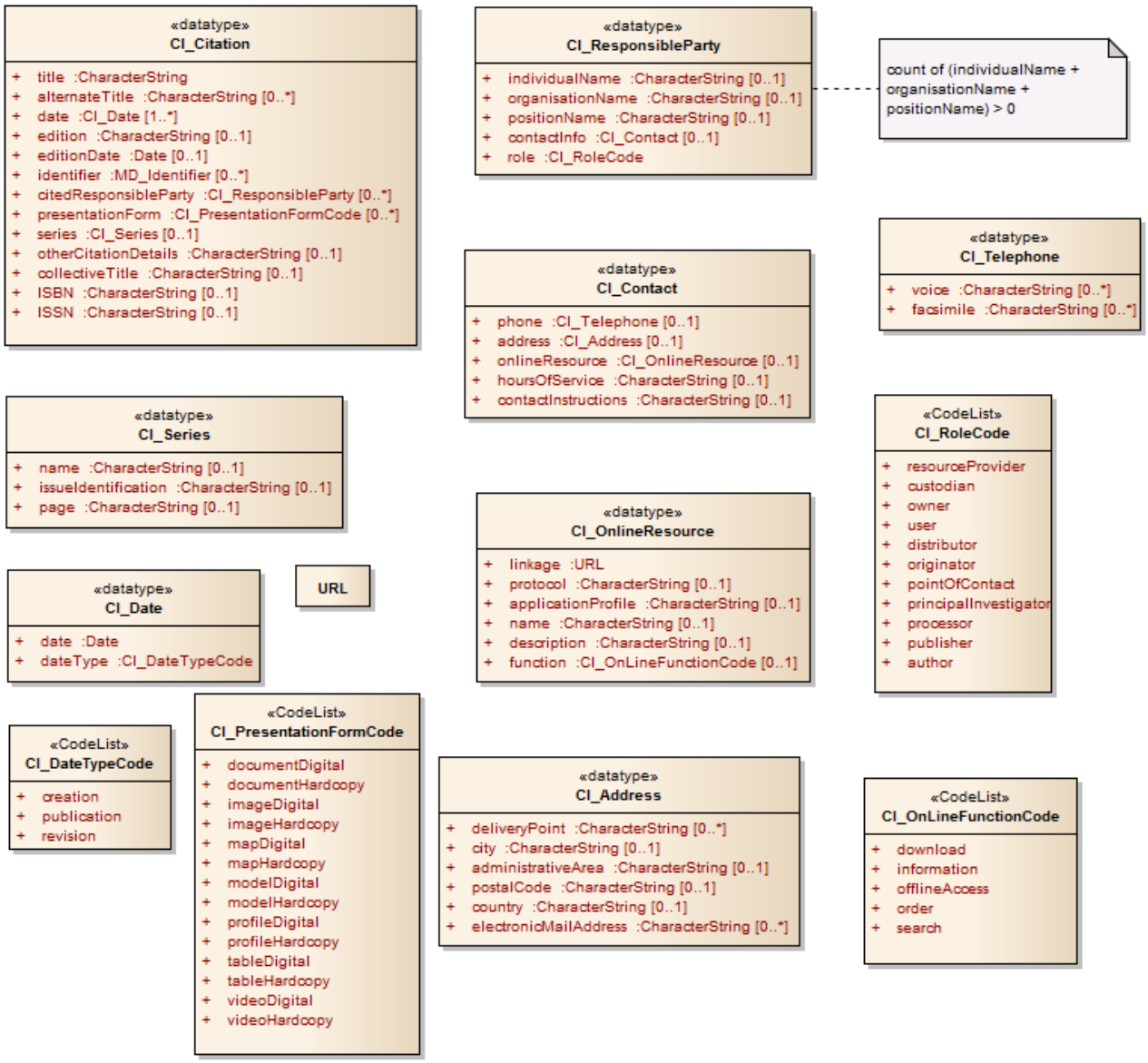


Abbildung 13: Klassendiagramm zu CI_Citation

ToDo 18

Es ist nicht geklärt wie wir mit solchen Rekursivschleifen im ogr-Modell umgehen sollen.

Eine Lösung wäre:

1. Zusätzliche Modellteile in XMI Format einlesbar machen
2. Einführung von Packetfiltern in conf Verzeichnis, auswählbar beim Laden um nicht alles von z.B. ISO 19115 einlesen zu müssen.
3. Beim Verfolgen des Modellbaumes an der Stelle aufhören wo ein Datentyp verwendet wird, der vorher schon mal im Pfad Verwendung fand. Attribute, die diesen Typ noch mal verwenden wollen auf text setzen.

Umsetzung:

Es wurde aus dem EA-Modell von INSPIRE das Packet „Data quality information“ aus ISO-19115 nach xmi exportiert und zusätzlich zum aaa_uml Schema eingelesen. Anschließend wurden einige Anpassungen vorgenommen, damit das aaa_uml Schema mit den Elementen aus dem ISO-Modell zusammenpassen.

```
UPDATE
      aaa_uml.datatypes
```

```
SET
      xmi_id = 'eaxmiid100'
WHERE
      xmi_id = 'eaxmiid0' AND
      name = '<undefined>';

UPDATE
      aaa_uml.uml_attributes
SET
      xmi_id = 'eaxmiid100'
WHERE
      xmi_id = 'eaxmiid0' AND
      id > 5000;

UPDATE
      aaa_uml.uml_attributes
SET
      classifier = 'EAID_5F5C5DEF_1901_496b_84EB_0756FFCD98A0',
      datatype = NULL
WHERE
      datatype = 'eaxmiid51';

UPDATE
      aaa_uml.uml_classes c
SET
      stereotype_id = s.xmi_id
FROM
      aaa_uml.stereotypes s
WHERE
      c.stereotype_id = '-1' AND
      s.name = 'DataType';
```

Es sind jedoch noch weitere Anpassungen notwendig, damit die Attribute gefunden werden und damit die weitere vertiefte komplexe Datenstruktur des Elementes DQ_Element.

Lösung:

Im Datenmodell werden die beiden Typen DQ_AbsoluteExternalPositionalAccuracy und DQ_RelativeExternalPositionalAccuracy als flacher DQ_Element Datentyp abgebildet. Dabei werden MD_Identifier, CI_Citation und DQ_Result als Text umgesetzt.

Eine weitere Verschachtelung kann später im Modell eingebaut werden. Ogr2ogr könnte in einer späteren Version die Inhalte der komplexen Typen in XML-Blob Form einlesen. Dann wäre die tieferen Elemente als strukturierter Text in den Attributen enthalten.

9.19 Frage 19

Frage:
AP_PTO.horizontaleAusrichtung bzw. vertikaleAusrichtung: Wie lassen sich die InitialValue zentrisch bzw. Mitte berücksichtigen?

Antwort:

Default-Werte werden im aktuellen Modell nicht berücksichtigt, da sie nicht im XMI-Dokument, welches aus EA exportiert wurde nicht enthalten sind.

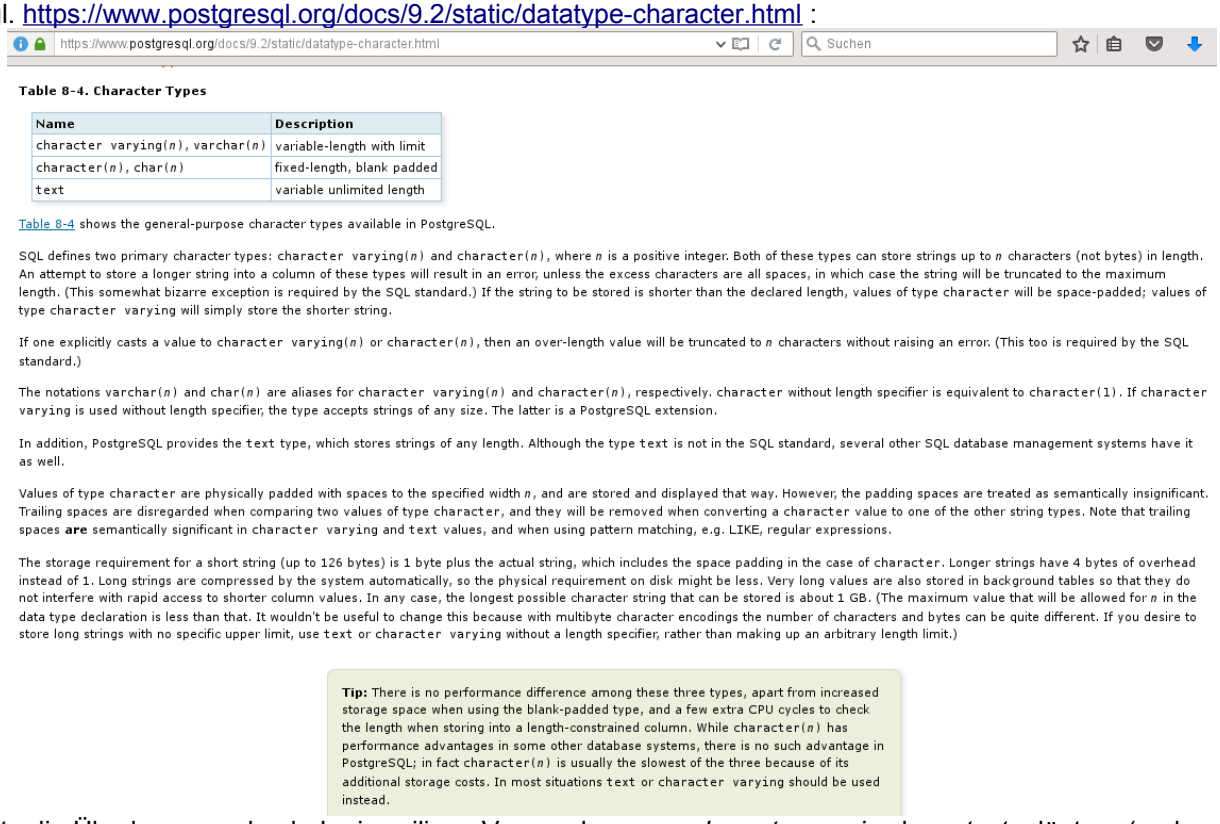
Default-Werte wären sicher auch nicht der richtige Ansatz, weil dabei Werte in der Datenbank erzeugt würden, die nicht den Daten im NAS-File entsprächen. Anders gesagt: Wenn das Attribut horizontaleAusrichtung in der NAS-Datei, die geladen werden soll, leer ist, bzw. fehlt sollte der Eintrag in der Tabelle des Objektes in der Datenbank auch leer bzw. NULL sein.

Wenn der Standard von den DHK richtig implementiert wurde, müssten InitialValues auch immer schon beim Erzeugen im Primärdatenbestand angelegt werden.

ogr2ogr liest aber für Sekundärbestände und in den einzulesenden NAS-Dateien sollten die InitialValues schon vorhanden sein.

9.20 Frage 20

Frage:
Vgl. https://www.postgresql.org/docs/9.2/static/datatype-character.html :



https://www.postgresql.org/docs/9.2/static/datatype-character.html

Table 8-4. Character Types

Name	Description
character varying(*n*), varchar(*n*)	variable-length with limit
character(*n*), char(*n*)	fixed-length, blank padded
text	variable unlimited length

Table 8-4 shows the general-purpose character types available in PostgreSQL.

SQL defines two primary character types: character varying(*n*) and character(*n*), where *n* is a positive integer. Both of these types can store strings up to *n* characters (not bytes) in length. An attempt to store a longer string into a column of these types will result in an error, unless the excess characters are all spaces, in which case the string will be truncated to the maximum length. (This somewhat bizarre exception is required by the SQL standard.) If the string to be stored is shorter than the declared length, values of type character will be space-padded; values of type character varying will simply store the shorter string.

If one explicitly casts a value to character varying(*n*) or character(*n*), then an over-length value will be truncated to *n* characters without raising an error. (This too is required by the SQL standard.)

The notations varchar(*n*) and char(*n*) are aliases for character varying(*n*) and character(*n*), respectively. character without length specifier is equivalent to character(1). If character varying is used without length specifier, the type accepts strings of any size. The latter is a PostgreSQL extension.

In addition, PostgreSQL provides the text type, which stores strings of any length. Although the type text is not in the SQL standard, several other SQL database management systems have it as well.

Values of type character are physically padded with spaces to the specified width *n*, and are stored and displayed that way. However, the padding spaces are treated as semantically insignificant. Trailing spaces are disregarded when comparing two values of type character, and they will be removed when converting a character value to one of the other string types. Note that trailing spaces **are** semantically significant in character varying and text values, and when using pattern matching, e.g. LIKE, regular expressions.

The storage requirement for a short string (up to 126 bytes) is 1 byte plus the actual string, which includes the space padding in the case of character. Longer strings have 4 bytes of overhead instead of 1. Long strings are compressed by the system automatically, so the physical requirement on disk might be less. Very long values are also stored in background tables so that they do not interfere with rapid access to shorter column values. In any case, the longest possible character string that can be stored is about 1 GB. (The maximum value that will be allowed for *n* in the data type declaration is less than that. It wouldn't be useful to change this because with multibyte character encodings the number of characters and bytes can be quite different. If you desire to store long strings with no specific upper limit, use text or character varying without a length specifier, rather than making up an arbitrary length limit.)

Tip: There is no performance difference among these three types, apart from increased storage space when using the blank-padded type, and a few extra CPU cycles to check the length when storing into a length-constrained column. While character(*n*) has performance advantages in some other database systems, there is no such advantage in PostgreSQL; in fact character(*n*) is usually the slowest of the three because of its additional storage costs. In most situations text or character varying should be used instead.

Bitte die Überlegungen bzgl. der jeweiligen Verwendung von *character varying* bzw. *text* erläutern (andere Character Types kommen nicht vor; character varying wird hier immer ohne Limit gesetzt).
Verwendungsbeispiele:
ap_pto.anlass = text[]
ap_pto.gml_id = text
ap_pto.identifier = character varying
ap_pto.schriftinhalt = character varying
oder auch
ax_punktortau.genauigkeitsstufe = character varying
ax_punktortau.genauigkeitswert = text

Antwort:

Wie schon in Frage 2 beantwortet, werden Enumeration Typen als character varying definiert, wenn die Werte der Aufzählung alphanumerisch sind und integer wenn es Zahlen sind.

CodeList Typen werden als Text umgesetzt, da hier Referenzen von Codelisten Registries reinkommen, von denen aber im UML-Modell nicht bekannt ist welche Werte es sind. Alle Attribute, die in UML als CharaterString definiert sind, werden im ogr-Schema als character varying ohne länge definiert, weil diese im

Modell auch nicht angegeben ist.

9.21 Frage 21

Frage:
ax_punktortau.genauigkeitsstufe = character varying – Warum hier nicht Integer?
ax_punktortau.vertrauenswuerdigkeit = character varying – Warum hier nicht Integer?

Antwort:

Das war in der Tat noch ein Fehler im Quellcode. Nun wird der richtige Typ ausgegeben.

9.22 Frage 22

Frage:
ax_georeferenziertegebaeudeadresse.hatauch text – Warum nicht NOT NULL, da das Modell zwingend Kardinalität 1 erfordert.

Antwort:

hatauch kann ich auf Anhieb im UML-Diagram nicht finden. Es geht um die Beziehung zwischen AX_GeoreferenzierteGebaeudeadresse und AX_LagebezeichnungMitHausnummer. Die ist laut UML-Diagramm AX_GeoreferenzierteGebaeudeadresse „weistAuf“ AX_LagebezeichnungMitHausnummer und AX_LagebezeichnungMitHausnummer „beziehtSichAuchAuf“ AX_GeoreferenzierteGebaeudeadresse, siehe Abbildung 14.

Laut AAA-Fachschema.xsd aus Version 6.0.1 vom 31.05.2009 die man unter http://www.adv-online.de/aaa-Modell/binarywriterservlet?imgUid=70425220-0746-2210-3ca0-c0608a438ad1&uBasVariant=11111111-1111-1111-1111-111111111111 beziehen kann, lautet die Verbindung zwischen AX_GeoreferenzierteGebaeudeadresse und AX_LagebezeichnungMitHausnummer aber „hatAuch“, siehe Abbildung 15.

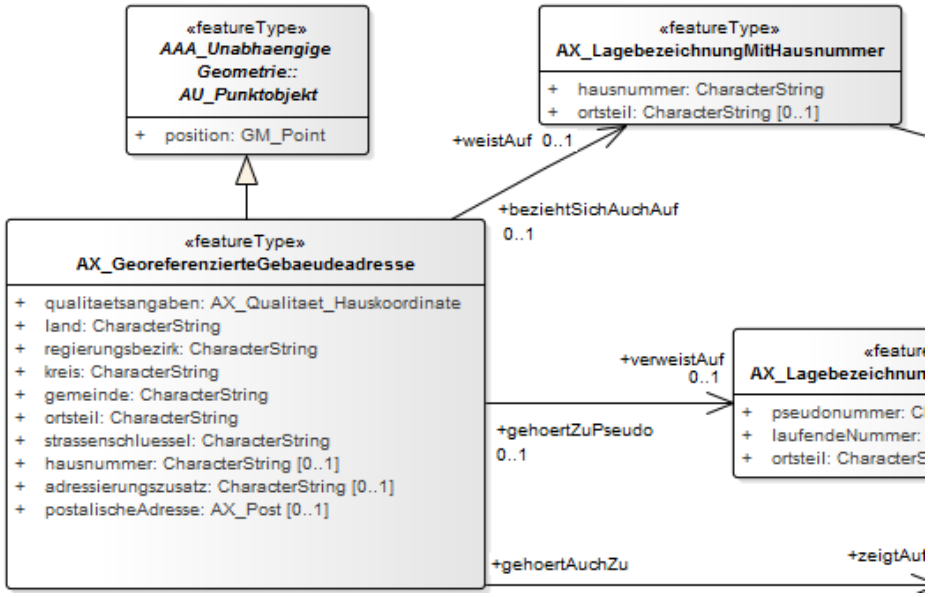


Abbildung 14: Beziehung zwischen AX_GeoreferenzierteGebaeudeadresse und AX_LagebezeichnungMitHausnummer

Daraufhin habe ich noch mal in die HTML-Ausgabe vom EA-Projekt geschaut und da findet man im UML-Modell „weistAuf“, siehe Abbildung 16, aber im davon abgeleiteten Implementierungsmodell „hatAuch“, siehe Abbildung 17.

```
<complexType name="AX_GeoreferenzierteGebaeudeadresseType">
  <complexContent>
    <extension base="adv:AU_PunktobjektType">
      <sequence>
        <element minOccurs="0" name="datensatznummer" type="string"/>
        <element name="qualitaetsangaben" type="adv:AX_Qualitaet_HauskoordinateType"/>
        <element name="land" type="string"/>
        <element name="regierungsbezirk" type="string"/>
        <element name="kreis" type="string"/>
        <element name="gemeinde" type="string"/>
        <element name="ortsteil" type="string"/>
        <element name="strassenschluessel" type="string"/>
        <element name="hausnummer" type="string"/>
        <element minOccurs="0" name="adressierungszusatz" type="string"/>
        <element minOccurs="0" name="postalischeAdresse" type="adv:AX_PostPropertyType"/>
        <element name="hatAuch" type="gml:ReferenceType">
          <annotation>
            <appinfo>
              <targetElement xmlns="http://www.opengis.net/gml/3.2">adv:AX_LagebezeichnungMitHausnummer</targetElement>
              <reversePropertyName xmlns="http://www.opengis.net/gml/3.2">adv:beziehtSichAuchAuf</reversePropertyName>
            </appinfo>
          </annotation>
        </element>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Abbildung 15: XSD Auszug zu AX_GeoreferenzierteGebaeudeadresse

Wenn man sich nun die Zeitstempel für Created und Modified ansieht, sieht man, dass die Klasse am 30.09.2009 erstellt wurde, was nach dem 31.05.2009 vom xsd ist, aber die Modified Zeitstempel sind unterschiedlich. Im UML-Modell wurde AX_LagebezeichnungMitHausnummer am 4.12.2014 geändert und im davon abgeleiteten Implementierungsmodell am 22.03.2010. „weistAuf“ sei demnach aktueller.

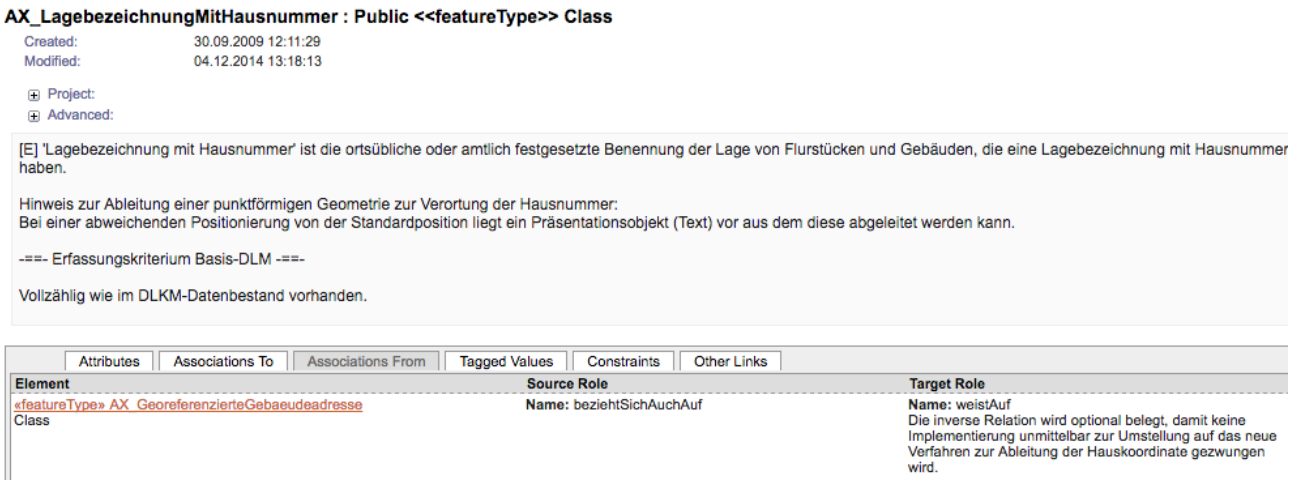


Abbildung 16: AX_LagebezeichnungMitHausnummer in UML-Modell

Merkwürdig ist jedoch, dass die aus dem EA-Projekt exportierte XMI-Datei das Assoziationsende „hatAuch“ nennt. Wir haben das also als „hatAuch“ umgesetzt.

Die Frage war aber warum nicht NOT NULL. Im Diagram steht 0..1 also optional und das steht auch in der Target Role, siehe Abbildung 16 und Abbildung 17.

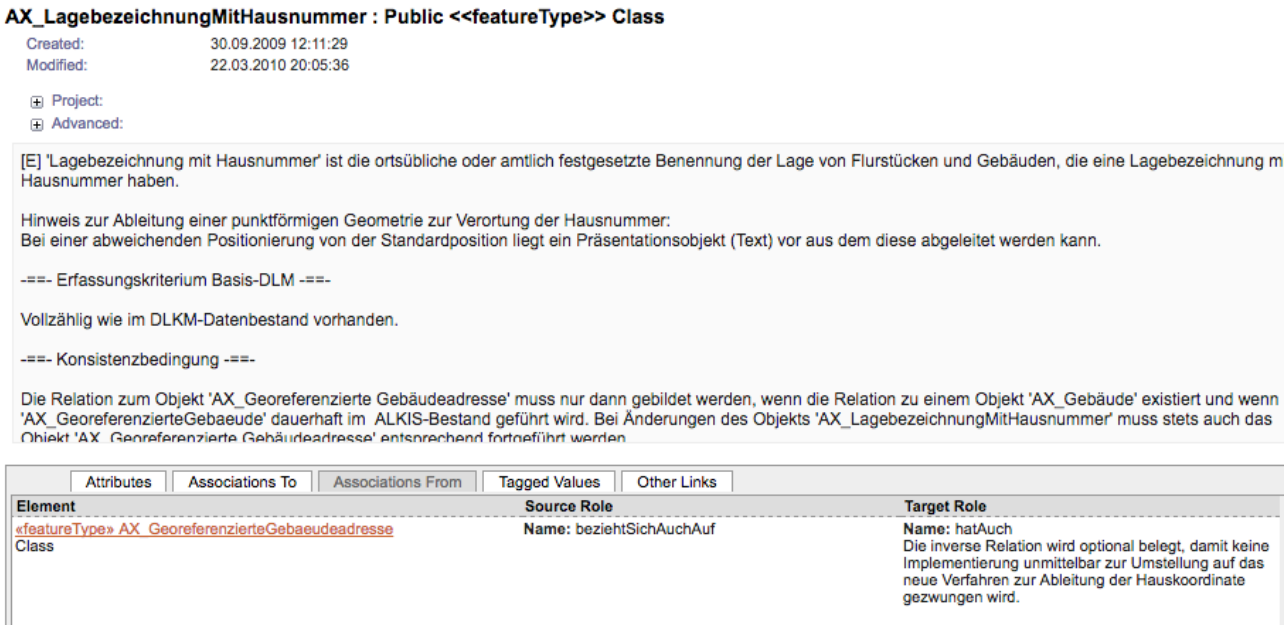


Abbildung 17: AX_LagebezeichnungMitHausnummer im Implementierungsmodell

Lösung:

hatAuch ist das richtige Assoziationsende und hat die Kardinalität 1, das ist jetzt auch so umgesetzt.

9.23 Frage 1, O. Schmidt

Frage:

Gemäß Ihrer Anleitung habe ich alle notwendigen Schritte durchgeführt, um eine Test-Datei zu importieren. Hierzu habe ich GDAL/OGR 2.2.0dev genommen, also die aktuellste verfügbare Version. Leider kommt nach dem Ausführen des Schrittes 6.4.1 der nachfolgende Fehler. Können Sie nachvollziehen, wieso der Import an dieser Stelle abbricht? Falls Sie eine Lösung dafür haben, teilen Sie mir bitte diese mit, damit ich mit den Tests fortfahren kann. Vielen Dank.

```
ogr2ogr -f "PostgreSQL" --config PG_USE_COPY NO -nlt CONVERT_TO_LINEAR
-append PG:"dbname=postnasneu active_schema=aaa_ogr user=postgres
host=localhost port=5432" -a_srs EPSG:25832 import_renamed.xml


ERROR 1: XML Parsing Error: comment or processing instruction expected at
line 8, column 1

FAILURE:
Unable to open datasource `import_renamed.xml' with the following drivers.
-> PCIDSK
-> PDF
-> ESRI Shapefile
-> MapInfo File
…
usw.
```

Antwort:

In Zeile 8 kommt in Ihrer umbenannten Datei ein Element vor obwohl das Dokument schon abgeschlossen war. Es handelte sich hier um einen Fehler im Umbenennungsskript, der behoben wurde.

In Ihrer XML-Datei war folgender Tag enthalten.

```
<erlaeuterung></erlaeuterung>
```

Im Umbenennungsskript wurde nun geprüft ob ein öffnendes Tag leeren Konten hat und wenn ja wurde es geschlossen. Der Parser kam dann aber auch noch an dem schließenden Tag an und führte noch mal ein Schließen des aktuellen Tags aus. Dieses mal das Wurzelelemnt. Daher sah das Ergebnis dann so aus.

```
...
 <erlaeuterung/>
</AX_NutzerbezogeneBestandsdatenaktualisierung_NBA>
<erfolgreich>true</erfolgreich>
```

Das heißt also das Dokument war beendet bevor es richtig angefangen hatte. Der Schließende Tag von AX_NutzerbezogeneBestandsdatenaktualisierung_NBA kam viel zu früh. Nach der Korrektur wird nicht mehr geprüft ob der Kontent des Tags leer ist und dann geschlossen, sondern geprüft ob es sich um ein leeres Element handelt. Die Methode empty_element? Liefert nur true wenn es sich um ein Tag der Form <tag/> handelt. Damit wird kein schließendes Tag erzeugt wenn die Form <tag></tag> gegeben ist, der Parser geht weiter und schließt es erst wenn er auf </tag> kommt. <tag/> wird hingegen sofort geschlossen.
Der Fehler ist damit behoben.

9.24 Frage 2, O. Schmidt

Frage:

wir haben nun mit der aktuellsten Version von xmi2db den Testdatensatz versucht zu importieren. Leider schlägt dies sofort mit der folgenden Fehlermeldung fehl. Die Verwendung von skipfailures führt dann dazu, dass der Importprozess zwar durchläuft, aber kein einziges Objekt importiert wird.

Wir könnten zur Lösung der Importprobleme Ihnen die Test-VM zur Verfügung stellen. Es handelt sich hierbei um eine VM im Oracle-VirtualBox-Format (.vdi). Darin enthalten wäre der Testdatensatz, die Installation von Postgres 9.4 und PostGIS 2.1 sowie der weiteren benötigten Pakete. Bitte teilen Sie uns mit, wie wir bzgl. der Fehlerbeseitigung weiter verfahren möchten.

root@schuldb:/postnas# ogr2ogr -f "PostgreSQL" --config PG_USE_COPY NO -nlt CONVERT_TO_LINEAR -append PG:"dbname=postnasneu active_schema=aaa_ogr user=postgres password=postgres host=localhost port=5432" -a_srs EPSG:25832 import_renamed.xml

ERROR 1: FEHLER: fehlerhafte Record-Konstante: »zentrisch«

LINE 1: ...21 17:39:41+00', ARRAY['DKKM1000'], 'ZAE_NEN', 0, 'zentrisch...

^

DETAIL: Linke Klammer fehlt.

ERROR 1: INSERT command for new feature failed.

FEHLER: fehlerhafte Record-Konstante: »zentrisch«

LINE 1: ...21 17:39:41+00', ARRAY['DKKM1000'], 'ZAE_NEN', 0, 'zentrisch...

^

DETAIL: Linke Klammer fehlt.

Command: INSERT INTO "ap_pto" ("position" , "identifier", "gml_id", "anlass", "beginnt", "advstandardmodell", "art", "fontsperrung", "horizontaleausrichtung", "schriftinhalt", "signaturnummer", "skalierung", "vertikaleausrichtung", "dientzurdarstellungvon") VALUES ('01010000004A0C02AB10111A414A0C0263F4DE5441'::GEOMETRY, 'urn:adv:oid:DERPLP110000E0dA', 'DERPLP110000E0dA', ARRAY['000000'], '2010/10/21 17:39:41+00', ARRAY['DKKM1000'], 'ZAE_NEN', 0, 'zentrisch', '571', '4111', 1, 'Basis', ARRAY['DERPLP110000DZZp']) RETURNING "ogc_fid"

ERROR 1: Unable to write feature 1 from layer AP_PTO.

ERROR 1: Terminating translation prematurely after failed

translation of layer AP_PTO (use -skipfailures to skip errors)

Antwort:

Dieser Fehler kam dadurch, weil die Enumerations noch nicht als varchar oder int umgesetzt waren. Das ist jetzt auch der Fall. Damit wird z.b. 'zentrisch' für den Datentyp varchar akzeptiert. Vorher war das ein Typ namens ap_horizontaleausrichtung, der ein Record verlangen würde.

9.25 Frage 1, M. Ambos

Frage:

1. Es gab Probleme mit Linestring (Eingabe) und erwartetem Multilinestring
Mögliche Behebung: ersetzen von MULTILINESTRING durch GEOMETRY ?

Antwort:

Gelöst mit Konstante LINESTRING_AS_GEOMETRY

9.26 Frage 2, M. Ambos

Frage:

Probleme mit „NOT NULL“
z. B. bei ax_gemarkungsteilflur ist die Vorgabe bei sonstigesmodell „NOT NULL“ und es treten deshalb Fehler beim Import auf.

Antwort:

Attribute von Union Typen werden jetzt auch immer auf nullable gesetzt.

9.27 Frage timestamp für Datetime in Postgres

Frage:

Im vorherigen Modell hatten Attribute wie beginnt oder endet Type Character Varying jetzt im neuen Modell datetime. Soll das so sein?

Antwort:

@M. Hentschel: Ich halte timestamp without timezone für die deutlich bessere Lösung, da sich SQL-Abfragen, die mit diesen Feldern hantieren, dann deutlich einfacher formulieren lassen.

9.28 Frage nur eine Geomertrie

Frage:

OGR liest zur Zeit nur eine Geometrie pro Tabelle ein. Kann objektkoordinaten rausgefiltert werden?

Antwort:

@M. Hentsche

Zu dem Problem, dass es nur ein Geometrie-Attribut geben darf: Die Objektkoordinate scheint mir nicht so elementar wichtig. Daher würde ich dafür plädieren, dass diese als Workaround entweder unterdrückt oder Typ character varying wird.

@S. Schliebner

Die genannten Vorkommen von objektkoordinaten sind bei uns nicht belegt, von daher

besteht unsererseits daran kein Bedarf.

@O. Schmidt

Sofern mit dem Typ "geometry" sämtliche Geometrietypen unabhängig vom eigentlichen Inhalt abgebildet werden können, würde ich auch für diese Vorgehensweise plädieren.

Lösung:

Fehlerd der Art „FEHLER: Geometry type (LineString) does not match column type (MultiLineString)“

treten seit der Einführung der Konstante LINESTRING_AS_GEOMETRY nicht mehr auf.

9.29 Frage Keine Fortführungen

Frage:

Wir wundern uns gerade, warum kein einziger endet-Zeitstempel im aaa_ogr Schema ankommt.

Antwort:

Es fehlt der Delete-Trigger, da dieser nicht direkt zum Datenmodell gehört.

Um das Modell aber mit Fortführungen befüllen zu können habe ich eine neue Version '22.11.2016 11:24' des Schemas unter

http://gdi-service.de/xmi2db/converter/db2ogr.php?umlSchema=aaa_uml&ogrSchema=aaa_ogr

zur Verfügung gestellt.

Das Datenschema selbst hat sich nicht geändert, aber ich habe am Ende noch

- die Tabellen delete sowie
- die Triggerfunktion delete_feature_hist und
- den Trigger delete_feature_trigger

angehängt.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

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