Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Semantische Überprüfung von CityGML-Daten auf Konformität zum Anwendungsbereich“

vorgelegt von

Dipl.-Geogr. (Univ.) Frank Stenger
U102523 , UNIGIS MSc Jahrgang 2013

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Gutachter:
Univ.-Prof. Dr. rer. nat. Thomas H. Kolbe

Kirchheim b. München, 30.05.2016

I. Danksagung

An erster Stelle danke ich Herrn Prof. Dr. Thomas H. Kolbe von der Technischen Universität München für die Chance, dieses Thema im Rahmen einer Masterarbeit bearbeiten zu können. Diese Arbeit wäre ohne Frau Tatjana Kutzner – ebenfalls von der Technischen Universität München – nicht zustande gekommen. Ebenfalls bedanken möchte ich mich für die wichtigen und aufschlussreichen Gespräche während und besonders zum Ende der Bearbeitungsphase. Der intensive Austausch in und nach der Abschlusspräsentation und das Feedback waren bei der Bearbeitung des Themas sehr hilfreich.

Bedanken möchte mich ich hiermit auch recht herzlich beim gesamten UNIGIS-Team für die Unterstützung während des Studiums.

Ich möchte mich auch bei meinen Vater für das Korrekturlesen der Arbeit bedanken. Mein großer Dank gilt meiner Familie und meiner Ehefrau Paola, deren Unterstützung ich immer hatte. Während der Zeit meines Studiums sind sie alle zu kurz gekommen, was sich nun ändern wird.

Zum Ende der Bearbeitungszeit ist meine Mutter verstorben. Besonderer Dank gilt ihr, da sie mir immer vermittelt hat, dass (Weiter-)Bildung sehr wichtig ist. Sie war zudem ein Vorbild, da sie selbst ein Studium neben dem Berufsleben aufnahm und mir so das lebenslange Lernen vorlebte.

II. Erklärung der eigenständigen Abfassung der Arbeit

Ich versichere, diese Masterthesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, welche wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Kirchheim b. München, den 30.05.2016 Frank Stenger

III. Zusammenfassung

3D-Stadtmodelle beinhalten neben den Geometrien auch Sachattribute, welche zusätzliche Informationen zu den Gebäuden bereitstellen. In diesem Fall spricht man von der Erweiterungsmöglichkeit um spezifische Anwendungsbereiche. Hierfür gibt es zwei unterschiedliche Methoden, die Application Domain Extension und die generischen Attribute.

In dieser Arbeit wird der Anwendungsbereich des AdV-CityGML-Profils vorgestellt, welche für das zukünftige AAA-Datenmodell von Wichtigkeit sein wird. Dieses AdV-CityGML-Profil basiert auf der Erweiterungsmöglichkeit durch generische Attribute und beinhaltet Vorgaben, welche die einzelnen Länder der Bundesrepublik Deutschland für die Abgabe ihrer CityGML-Daten beachten müssen. Diese Vorgaben sind in einem Prüfplan zusammengefasst.

In der vorliegenden Arbeit werden zum Teil diese Vorgaben in der Modellebene in das UML-Datenmodell mittels OCL integriert und einige auf Datenebene modelliert, um letztlich einen Großteil dieser Vorgaben umzusetzen. Das Ziel ist die Umsetzung des Prüfplans in eine Prüfsoftware mit Hilfe unterschiedlichster Softwarewerkzeuge wie Python und FME von Safe Software. Anhand von AdV-CityGML-Referenzdatensätzen wird die Funktionalität dieser Software evaluiert und eine Prüfung von CityGML-Datensätzen nach Vorgaben der AdV vorgenommen. Die Software prüft die CityGML Detaillierungsgrade LOD1 und LOD2, wenn diese als MultiPart modelliert wurden und gibt das Ergebnis als Prüfprotokoll aus.

IV. Abstract

3D city models include besides the geometries attributes, which provide additional information on the buildings. In this case, one speaks of the expandability to specific application areas. There are two different methods, the Application Domain Extension and the generic attributes.

In this work, the AdV-CityGML profile is presented, which is important for the future AAA data model. This AdV-CityGML profile includes requirements which the States of Federal Republic of Germany need to consider to provide their CityGML data. These requirements are summarized in a quality control plan.

In the present work a part of these requirements are integrated in the UML data model using OCL and some are modeled on data level to ultimately implement a large part of these requirements.
The goal is the implementation of the quality control plan in a test software with various software tools such as Python and FME from Safe Software. Using AdV-CityGML reference data sets the functionality of this software will be evaluated and a test of CityGML data according to the specifications of AdV will be done. The software checks CityGML Level of Detail LOD1 and LOD2 when modeled as MultiPart and generate a report.

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung und thematischer Kontext

Das CityGML-Datenmodell ist ein internationaler Standard für 3D-Stadtmodelle. CityGML ist einerseits ein semantisches Modell, das durch ein formales Datenmodell beschrieben wird und andererseits ein Austauschformat für virtuelle 3D-Stadt- und Landschaftsmodelle (Kolbe 2009). CityGML basiert auf GML3, wurde 2008 zum OGC-Standard für die 3D-Stadtmodellierung erhoben und ist seit 2012 in der Version 2.0 verfügbar. Für die Modellierung grafischer Objekte wird dabei auf die ISO 19100 Standardfamilie zurückgegriffen. Entsprechend der ISO 19109 werden geografische Merkmale als Abstraktionen von realen Objekten angesehen und als Klassen nach UML-Notation modelliert. Jedes Objekt kann damit eine Sammlung an räumlichen und nicht-räumlichen Attributen besitzen und objektorientiert als Akkumulation geografischer Objekte umgesetzt werden (Kolbe 2009).
Das CityGML-Modell wird seit seiner Entwicklung für wissenschaftliche Untersuchungen verwendet. Dabei spielen sowohl die einzelnen Gebäude als auch die Modellierung von Stadtteilen oder ganzen Städten eine wesentliche Rolle.
Das CityGML-Modul Building weist neben Geometrien Sachattribute auf, welche zusätzliche Informationen zu den Gebäuden bereitstellen. Durch unterschiedliche Detailierungsgrade (Level of Detail) steigt deren Anzahl, wodurch das CityGML-Modul Building realitätsnäher wird. Durch die Beschreibung mit Hilfe von Merkmalen, die je nach Anwendungsbereich relevant sein können, wird deren Überprüfung sinnvoll. Das CityGML-Modell Building kann somit neben Geometrien auch auf Attributwerte hin überprüft werden, die für bestimmte Anwendungsbereiche notwendig sind. Ein Prüfverfahren hinsichtlich der Geometrien wurde bereits von Coors (Wagner et al. 2013) erfolgreich beschrieben und umgesetzt.

1.2 Motivation und Problemstellung

“Quality is conformance to requirements”. Dieser Satz stammt von Philip B. Crosby (1979) und bedeutet „Qualität ist die Erfüllung von Anforderungen“. Dieser Satz zeigt die Komplexität und Herausforderung eines Projektes. Wenn die Anforderungen genau formuliert und spezifiziert sind, kann die Erfüllung dieser Anforderungen mit entsprechenden Werkzeugen genauestens ermittelt werden (Löwner et al. 2013).

Das Thema 3D-Stadtmodelle wird durch die Möglichkeit, diese thematisch durch spezifische Informationen zu erweitern, für immer mehr Anwendungsbereiche interessant. Anfangs wurden mit den Geometrien innerhalb des CityGML-Modells Analysen bezüglich des Katastrophenschutzes, Lärmbelästigung und Sichtbarkeitsanalysen durchgeführt. Nun rückt auch das Gebäude mit seinen Attributen in den Mittelpunkt, wenn es z.B. um die Spezifikation der Vermessungsverwaltungen der Länder und das daraus resultierende AdV-CityGML-Profil geht. Um das CityGML-Datenmodell gemäß der Vorgaben der AdV zu validieren, ist folglich die Überprüfung dieser Attribute notwendig.
Die Verwendung des CityGML-Datenmodells hinsichtlich eines Anwendungsbereichs, bei denen das Gebäude mit seinen Attributen in den Mittelpunkt rückt, ist in der Geoinformatik ein hochaktuelles Thema, das auch in Zukunft mehr und mehr an Bedeutung gewinnen wird. Mit der Energy ADE oder Noise ADE wurden bereits zwei Anwendungsbereiche umgesetzt. Das Thema 3D-Stadtmodelle wird durch die Erweiterung des CityGML-Datenmodells mittels generischer Attribute oder auch Application Domain Extensions (ADE) für immer mehr Anwendungsbereiche interessant, da diese spezifischen Informationen implementiert werden können. Die Gebäudemodelle können dann für weitere Analysen verwendet werden.
Es lassen sich weitere Anwendungsbereiche modellieren und CityGML-Datensätze auf diese Spezifikationen hin überprüfen.
Neben der Definition des Anwendungsbereiches sind CityGML-Daten notwendig, die diesem Anwendungsbereich entsprechen. Das zentrale Thema dieser Arbeit geht der Frage nach, ob vorhandene CityGML-Daten sich für diese Anwendung eignen und ob alle für diese Anwendung relevanten Attribute vorhanden sind.

In dieser Arbeit sollen die Anforderungen der AdV in das CityGML-UML-Modell integriert werden. Die Umsetzung der gültigen Attributwerte soll durch OCL-Regeln realisiert werden. Neben der Erstellung des AdV-CityGML-Profils auf Modellebene soll in dieser Arbeit ein Prüfverfahren erarbeitet werden, das sich der Validierung von CityGML-Datensätzen nach Vorgaben des AdV-CityGML-Profils widmet. Die CityGML-Daten sollen anhand eines Prüfschemas hinsichtlich der Vollständigkeit der Attribute und dem Vorhandensein von Attributwerten für diesen Anwendungsbereich geprüft und unvollständige oder fehlerhafte Datensätze in einem Prüfbericht ausgegeben werden. Hierbei soll auch auf den Unterschied zwischen den Level of Detail (LOD) 1 und 2 und die daraus resultierenden prüfbaren Attribute eingegangen werden.

Zentrale Frage der Arbeit ist die Modellierung der Anforderungen des Anwendungsbereichs mittels einer geeigneten Sprache, der Einschränkung der möglichen Wertebereiche und schlussendlich die Überprüfbarkeit des AdV-CityGML-Modells hinsichtlich der Semantik. In den drei Anwendungsfällen werden die Beispieldaten der AdV sowie die CityGML-Daten des LDBV Bayern und Thüringen hinsichtlich der Vorgaben überprüft und eine Aussage bezüglich der Konformität der Datensätze und der Korrektheit der Prüfsoftware getroffen.

In Abbildung 1 ist das schematische Prüfverfahren, welches in dieser Arbeit realisiert werden soll, veranschaulicht.

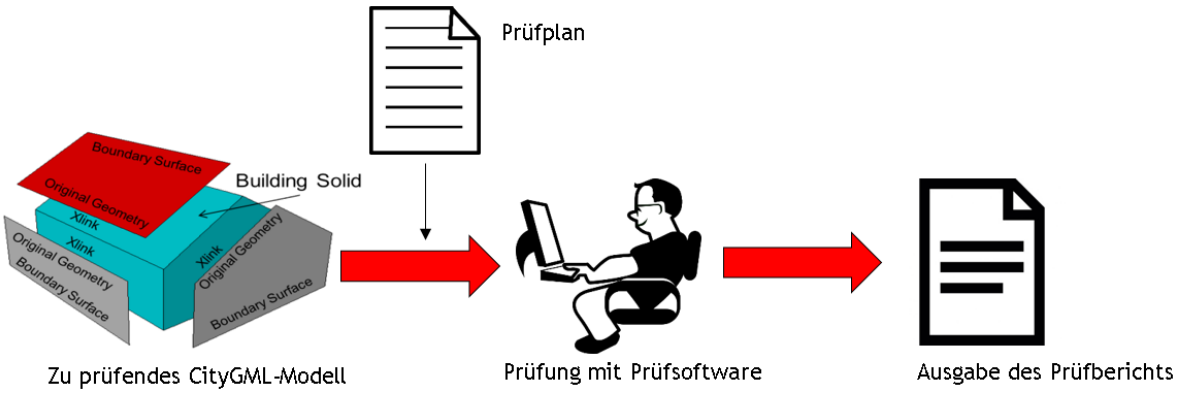

Abbildung 1: Schematisches Prüfverfahren

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in sieben Kapitel.

Im ersten Kapitel wird eine kurze Einführung des Themas der Arbeit gegeben.

Im zweiten Kapitel werden die in der Einleitung erwähnten Punkte aufgegriffen und in einen größeren Zusammenhang zur Thematik gebracht und weiter ausgeführt. Hierfür werden die relevanten Standards näher erklärt und um die angewandten Methoden zur Umsetzung des AdV-Profils aufgezeigt.

Das dritte Kapitel widmet sich den verwendeten Softwarewerkzeugen zur Modellierung von Datenmodellen, der Programmiersprache Python und des Entwicklungswerkzeuges FME und geht dabei genauer auf die interne Sprache von FME ein.

Im vierten Kapitel geht es um das Konzept der semantischen Konformitätsprüfung. Es wird die Umsetzung auf Modellebene und die Umsetzbarkeit auf der Datenebene von Attributen und Attributwerten behandelt.

Das fünfte Kapitel umfasst die prototypische Implementierung. Dabei wird auf die FME-Workspaces und das verwendete Vorlagenkonzept näher eingegangen. Anschließend wird die Programmoberfläche näher beschrieben und der Konvertierungsprozess der Software ausführlich erklärt.

Im sechsten Kapitel werden in einer Evaluierung mit drei Testfällen die Software und CityGML-Daten geprüft. Dabei wird mit manipulierten Referenzdaten die Software geprüft und mit zwei Testfällen die Umsetzung des AdV-CityGML-Profils evaluiert.

Das letzte Kapitel befasst sich mit der Umsetzung der Vorgaben innerhalb der entwickelten Prüfsoftware, fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf die Weiterentwicklung der Thematik.

1.4 Verwandte wissenschaftliche Arbeiten

Dieses Kapitel gibt einen Überblick über verschiedene Arbeiten, die sich mit der in dieser Arbeit behandelten Thematik auseinandersetzen.

In der Masterarbeit *Energiebedarfsanalyse urbaner Räume* aus dem Jahr 2013 erläutert Banfi die Erweiterung des CityGML-Standards mit Hilfe einer ADE. Die in dieser Arbeit verwendete Energy ADE ist eine Erweiterungsmöglichkeit des CityGML-Profils um einen Anwendungsbereich, die sich in diesem Fall auf das Building-Modul bezieht [(Banfi 2013)].

Gozdz, K. et al. behandeln in *The possibilities of using CityGML for 3D representation of buildings in the cadastre* das Thema CityGML, welches für den Aufbau eines 3D Katasters mittels Application Domain Extension (ADE) als Grundlage dient. Analog zur Arbeit von Banfi wird ein tieferer Einblick in das Thema des Aufbaus eines Anwendungsbereichs und die Erweiterung des CityGML-Modells aufgezeigt [(Gozdz et al. 2011)].

In *Formalization of spatial constraints* beschreibt Werder erste Ansätze anderer Arbeiten hinsichtlich der Verwendung der Object Constraint Language für geografische Daten. Als prominentestes Projekt geht er auf das AFIS-ALKIS-ATKIS-Referenzmodell (AAA) der GDI-DE ein, dessen Anwendungsschema den Standards der ISO als auch der OGC entspricht, in der Unified Modeling Language modelliert wurde und das Ziel der Interoperabilität geografischer Daten verfolgt. Der Vorschlag des Begriffes Geo Object Constraint Language (GeoOCL) wird angeschnitten und erste Beispiele hierfür angebracht, aber auch explizit darauf hingewiesen, dass zu diesem Thema mehr geforscht werden muss, um Einschränkungen in XML-Schemas abbilden zu können. Dafür ist eine Erweiterung der OCL Grammatik um den neuen Datentyp *Geometry* notwendig [(Werder 2009)].

In *Ensuring quality of geographic data with UML and OCL* beschreiben Casanova et al. die Möglichkeit der Verwendung von OCL zur Qualitätssicherung von geografischen Daten in Form von TeleAtlas Straßengraphen hinsichtlich der erlaubten Fahrtrichtungen. Diese OCL-Konstrukte wurden in das UML-Modell eingebracht und vervollständigen diese Klassendiagramme [(Casanova, M., Wallet, T., & D'Hondt, M. 2000)].

In *Spezifikation von Prüfplänen und Prüfergebnissen zur Validierung von 3D-Stadtmodellen* aus dem Jahr 2014 beschreiben Wagner et al. den Aufbau und die Spezifikation eines Prüfplanes, welcher anwendungsspezifisch zum Beispiel Volumenkörper hinsichtlich Geschlossenheit oder hinsichtlich bestimmter Geometrietypen prüfen soll. Als Forschungsergebnis ist die Software CityDoctor realisiert worden, welche den spezifizierten Prüfplan zur Validierung von CityGML-Datensätzen anwendet [(Wagner et al. 2014)].

Die wissenschaftliche Arbeit *Geometric-semantical consistency of CityGML models* behandelt die Überprüfung von CityGML-Daten bezüglich Geometrie und Semantik. Darin wird die Notwendigkeit der Qualitätssicherung angesprochen, welche für 3D-Stadtmodelle notwendig ist. Die Modellierungsart Solid oder MultiSurface für das Modul Building stellt dabei ein erstes Kriterium der Überprüfung dar. Über die Definition, welche Geometrie valide ist, werden Prüfroutinen erarbeitet. Die semantische Überprüfung dieser Geometrien wird mit der Syntax der Objekt Constraint Language vorgenommen [(Wagner et al. 2013)].

2 Grundlagen zur Modellierung von 3D-Stadtmodellen

2.1 Theoretische Grundlagen der Datenmodellierung mit Standards der OMG

In dem nachfolgenden Kapitel werden die theoretischen Grundlagen des 3D-Stadtmodells behandelt. Im ersten Teil wird auf die Modellierungssprache Unified Modeling Language eingegangen und anschließend erklärt, in welcher Weise die Einschränkungen mit Hilfe der Object Constraint Language implementiert werden. Im darauffolgenden Abschnitt wird erklärt, wie das UML-Modell in eine maschinenlesbare Form exportiert wird.
Die *Object Management Group* (OMG) wurde zur Entwicklung von Standards für die herstellerunabhängige systemübergreifende objektorientierte Programmierung gegründet. Die bekanntesten Standards, welche durch die OMG entwickelt wurden sind MOF (Meta Object Facility), XMI (XML Metadata Interchange), UML (Unified Modeling Language), sowie OCL (Object Constraint Language) (OMG 2014b, 2014a, 2015).
Die drei letztgenannten sind für diese Arbeit relevant und werden in den nachfolgenden Abschnitten näher erklärt.

2.1.1 UML – Unified Modeling Language

Die Unified Modeling Language definiert eine allgemein verwendbare Modellierungssprache, dessen Einsatzgebiet sich nicht nur auf Softwareentwicklung beschränkt. Sie stellt zudem Diagramme und Notationsgebiete (einzelne Bestandteile der Diagramme) zur Verfügung, mit denen statische als auch dynamische Aspekte unterschiedlichster Anwendungsgebiete modelliert werden können.
Die Unified Modeling Language bietet vielerlei Vorteile bei der Modellierung und zeichnet sich durch Eindeutigkeit und durch eine präzise Semantik der Notationselemente aus. Selbst bei komplexen Diagrammen erleichtern die grafisch visualisierten Notationselemente das Verständnis. Hierdurch werden differenzierte Sichtweisen ermöglicht.
In der UML gibt es mehrere Diagrammtypen. Diese können auf der obersten Ebene in Strukturdiagramme und Verhaltensdiagramme unterteilt werden. Während die Strukturdiagramme

statische, zeitunabhängige Elemente des Systems modellieren, werden mit den Verhaltensdiagrammen dynamische Aspekte, das Verhalten des Systems und seiner Komponenten modelliert. Klassendiagramme beinhalten statische Strukturbestandteile eines Systems, sowie deren Eigenschaften und Beziehungen. Es dient dabei als eine Art Bauplan für Objekte (Kecher 2009).

Die UML ist eine Antwort auf die Herausforderungen heutiger komplexer Systeme, deckt ein breites Spektrum von Anwendungsgebieten ab und eignet sich für konkurrierende, verteilte, zeitkritische und sozial eingebettete Systeme (Oestereich und Scheithauer 2014).

Als plattformunabhängige Standardsprache zur Darstellung und Erstellung komplexer Systeme wird sie zur Modellierung von objektorientierten Anwendungen verwendet.

Die Modellierung erfolgt dabei in Form von Diagrammen, die entweder ein statisches oder ein dynamisches Modell beschreiben. Die in dieser Arbeit verwendeten Diagrammtypen sind Klassendiagramme.

In einem Klassendiagramm werden unterschiedliche Klassen, sowie deren Beziehungen untereinander dargestellt. Klassendiagramme werden allgemein zur Modellierung von Datenmodellen verwendet. Aktivitätsdiagramme hingegen dienen der Modellierung von Prozessen.

Klassen bilden das zentrale Konzept von Klassendiagrammen und stellen eine Abstraktion der Objekte der realen Welt dar. Die Modellierung von Geodaten kann auf verschiedene Weise geschehen. Einerseits auf der Schicht der Datentransferformate und andererseits auf der konzeptuellen Ebene, welche formatunabhängig ist. Wenn die Erstellung von Modellen auf konzeptueller Ebene erfolgt, werden sogenannte konzeptuelle Modellierungssprachen verwendet. Die AdV hat sich zur Beschreibung des AAA-Modells und der Objektartenkataloge entschieden, die Datenmodellierungssprache Unified Modeling Language (UML) zu verwenden. Wenn ein Modell mittels einer Modellierungssprache niedergeschrieben wird, so wird auch von einem konzeptuellem Schema gesprochen (Kutzner et al. 2012).

Diese wird auch von ISO/TC 211 im Bereich der Normung von Geoinformationen verwendet und ist in diesem Bereich die gängige Modellierungs- und Beschreibungssprache. Um innerhalb der Normfamilie 19100 eine einheitliche Nutzung der UML zu gewährleisten, hat man deren Verwendung in der ISO-Spezifikation 19103 Conceptual Schema Language definiert.

Eine formale Sprache als Beschreibungssprache bewirkt eine vollständige und klar interpretierbare Beschreibung von Inhalt und Struktur der Geodatenbestände. Die Anwendungsschemata können von anderen Modellierungsprogrammen interpretiert und in interne Datenstrukturen bzw. Datenbankstrukturen übersetzt werden. Als Auszeichnungssprache wird das XML-Format des World-Wide-Web Consortiums (W3C) verwendet (AdV 2015c).

2.1.2 OCL – Object Constraint Language

Die Object Constraint Language ist eine einfache formale Sprache, mit welcher UML-Modellen weitere Semantik hinzugefügt werden kann, welche mit den übrigen UML-Elementen nicht oder nur umständlich ausgedrückt werden könnte. Der OCL-Formalismus hat große Ähnlichkeit mit der Sprache Smalltalk sowie mit der Prädikatenlogik (Störrle 2005; Clark und Warmer 2002).
Laut Louwsma et al. (2006) können OCL- Ausdrücke in UML-Modellen Werte spezifizieren. Diese Werte heißen Invarianten und müssen zu allen Zeitpunkten gültig sein. Die Standard-OCL-Datentypen sind *integer*, *real*, *string* und *boolean* (Louwsma et al. 2006).
OCL-Ausdrücke werden durch die Angabe des Kontextes eingeleitet:
Bsp.: Invarianten (OMG 2014a)

context *Company*
  inv*: self.manager.isUnemployed = false*
  inv*: self.employee->notEmpty()*

Mit Hilfe von „self“ greift man auf eine spezifische Instanz des Kontextes und seine Eigenschaften zu. In dem Beispiel wird die Verwendung des OCL-Ausdrucks für die Invariantenzusicherung („inv“) gezeigt, was eine Einschränkung für eine bestimmte Klasse darstellt. Diese Einschränkung muss zu jeder Zeit Gültigkeit haben (Oestereich und Bremer 2012).
Neben *Invarianten* gibt es *Collections*, *Sets*, *OrderedSets*, *Bags* und *Sequences*, welche vordefinierte Typen in OCL sind. Diese können eine größere Anzahl von vordefinierten Operationen nach sich ziehen. Eine Eigenschaft der *collection* ist, dass diese durch einen Pfeil ‘->’ auf sich selbst zugreift, gefolgt von dem Namen der Eigenschaft. Die Kurznotation für die

collect-Operation ist die dot-Notation, das heißt für *self->collect(participants)*, wird *self.participants* geschrieben (OMG 2014a). Das folgende Beispiel zeigt dies anhand des Kontextes *person*.

```
context Person inv
  self.employer->size() < 3
```

Dies gilt für die Eigenschaftsgröße des *Sets self.employer*, welche die Anzahl der Arbeitgeber der Person selbst führt.

```
context Person inv
  self.employer->isEmpty()
```

Dies betrifft die *isEmpty* Eigenschaft auf dem *Set self.employer*. Es wertet es als wahr, wenn die Menge der Arbeitgeber sonst leer und falsch ist (OMG 2014a).
Eine andere Art von Einschränkungen sind die Vor- und Nachbedingungen. Vorbedingungen geben Eigenschaften von Eingabeparametern einer Operation an, welche vor Ausführung der Operation erfüllt sein müssen, sonst würde die Ausführung der Operation scheitern. Nachbedingungen geben Eigenschaften von Ausgabeparametern einer Operation oder globalen Variablen an, die nach der Operationsausführung gelten müssen. Dieses Methodenergebnis wird in der Nachbedingung mit *result* bezeichnet. Nach den Schlüsselwörtern *pre* und *post* folgt die jeweilige Vor- und Nachbedingung (Rumpe 2012).
Im Beispiel kann ein Hotelgast nur ein Bad verwenden, wenn dieses zu einem Zimmer gehört oder sich auf der gleichen Etage befindet. Diese Regel wird in der Vorbedingung für das Bad verwendet (Clark und Warmer 2002).

```
context Bathroom::uses(g : Guest)
pre:  if room->notEmpty then
          room.guests->includes(g)
      else
          g.room.floorNumber = self.floorNumber
      endif
post: usage = usage@pre + 1
```

Es gibt eine starke Verbindung zwischen den Standards der OMG. Die bekannteste Initiative der OMG in die Model Driven Architecture. Kernpunkt der MDA besteht in dem Ansatz, dass Modelle die Basis für die Softwareentwicklung sind. Aus diesem Grund sollten Modelle gut, solide, konsistent und kohärent sein, was mit Hilfe der UML und OCL ermöglicht werden kann (Warmer und Kleppe 2003).

2.1.3 XMI – XML Metadata Interchange

Die XML Metadata Interchange (XMI) ist ein weiterer Standard der Object Management Group (OMG) und ein Austauschformat zwischen Software-Entwicklungswerkzeugen.
Wie alle anderen Standards der OMG ist das Format offen, anbieterneutral und bietet dadurch den Datenaustausch von Objekten auf der Basis von Meta-Metamodellen nach der Meta-Object Facility (MOF), definiert im ISO/IEC 19508 Standard. Es definiert die folgenden Aspekte bei der Beschreibung der Objekte in XML:

- Darstellung von Objekten in Form von XML-Elementen und Attributen.
- Standard-Mechanismen, um Objekte in derselben Datei oder in querverweisenden Dateien zu verknüpfen
- Validierung von XMI-Dokumenten mit Hilfe von XML-Schemas

XMI liefert Lösungen für die oben genannten Probleme durch die Angabe von EBNF Produktionsregeln für das Erstellen von XML-Dokumenten und Schemata, um Objekte konsistent zu verteilen (OMG 2014b). XMI basiert auf der Extensible Markup Language (XML), welches das universelle Format darstellt, um MOF-basierte Modelle auszutauschen. XML wiederum ist aber nicht objektorientiert, sondern definiert lediglich XML Elemente und XML Attribute. Durch den Export des AdV-CityGML-Modells nach XMI kann das Modell mit Hilfe von Python weiterverarbeitet werden.

2.2 CityGML als Modellierungsstandard von 3D-Stadtmodellen

CityGML ist ein offenes Datenmodell und ein internationaler Standard des *Open Geospatial Consortiums* (OGC) für die Darstellung und den Austausch von 3D-Stadtmodellen. CityGML wurde von der Special Interest Group 3D (SIG3D) der Initiativen GDI-NRW (bis 2010) und GDI-DE (ab 2011) sowie der OGC CityGML Standards Working Group (SWG) spezifiziert (Kolbe et al. 2014). Die SIG3D ist verantwortlich für Qualitätsanforderungen, Modellierungsregeln und der Koordination der Weiterentwicklung von 3D-Stadtmodellen (SIG3D Online 2016).

Das Datenmodell definiert dreidimensionale Geometrien, Topologien, Semantik und die Darstellung topografischer Objekte. CityGML basiert auf der OGC *Geography Markup Language* (GML), welche ein standardisiertes Geometriemodell bereitstellt. Durch die sehr gut definierte Semantik dieses Datenmodells ist der interoperable Datenaustausch von CityGML hinsichtlich der Beziehung von Webdiensten und Geodateninfrastrukturen vereinfacht.

CityGML hat sich seit seiner Einführung im Jahr 2008 weltweit immer mehr durchgesetzt und seitdem ein weltweiter Standard. Softwarehersteller als auch Anwendungen bauen auf diesen Standard (Gröger und Plümer 2012). Im Jahr 2012 ist Version 2.0 veröffentlicht worden und wird auch für die nächsten Jahre valide sein (Kolbe et al. 2014).

Für die Visualisierung von Stadt- und Bauleitplanung, der Planung von Funknetzen, sowie für den Katastrophenschutz werden 3D-Stadtmodelle benötigt. Diese hochgenauen 3D-Stadtmodelle bestehen aus einem Geländemodell, 3D-Gebäudedaten, Verkehrswegen und vielen weiteren Informationen (Gröger et al. 2005).

2.2.1 Konzept des Level of Detail

CityGML gibt es in unterschiedlichen Detailierungsgraden. Dies stellt ein zentrales Konzept der Modellierung von CityGML dar. Das Konzept der Level of Detail (LOD) ermöglicht sowohl eine differenzierte Verfeinerung der geometrischen Ausprägung von Objekten als auch die Möglichkeit, weitere semantische Eigenschaften dem Modell hinzuzufügen.

Mit der Einführung von CityGML 2.0 wurde die geometrische Repräsentation für Gebäude um den LOD0 erweitert. Somit gibt es insgesamt fünf LOD im Gebäudemodul, die in Abbildung 2 dargestellt sind (Löwner et al. 2012).

Der LOD0 ist ein Regionalmodell, das aus einem Digitalen Geländemodell/Gebäudegrundriss besteht, auf das ggf. ein Satelliten-/Luftbild oder eine Karte gelegt wird und keine semantische Strukturierung hinsichtlich Begrenzungsflächen aufweist.

Der LOD1 ist das bekannte Klötzchenmodell, welches weder Dachstrukturen noch Texturen enthält. LOD1 weist ebenfalls keine semantische Strukturierung hinsichtlich Begrenzungsflächen auf.

Der LOD2 beinhaltet eine generalisierte Abbildung der Gebäude-Außenkontur und eine semantische Strukturierung.

Abbildung 3 zeigt einen Teilausschnitt des 3D-Stadtmodells von Pullach i. Isartal im CityGML-Format im sogenannten LOD2 mit Informationen zu den Dachstrukturen.

Im Architekturmodell LOD3 sind differenzierte Dach- und Fassadenstrukturen geometrisch ausgeprägt. Darüber hinaus weist das Gebäude Öffnungen wie Türen und Fenster auf.

Der LOD4, als höchster Detaillierungsgrad, zeigt Strukturen im Gebäudeinneren des Modells.

Zu jedem LOD werden entsprechende Erfassungskriterien und Genauigkeitsanforderungen angegeben. Die Detaillierungsgrade LOD0 und LOD4 sind Ergänzungen der SIG 3D, wohingegen LOD1 bis 3 bereits vorher Erwähnung in der Literatur (z.B. Köninger und Bartel 1998) fand (Gröger et al. 2005).

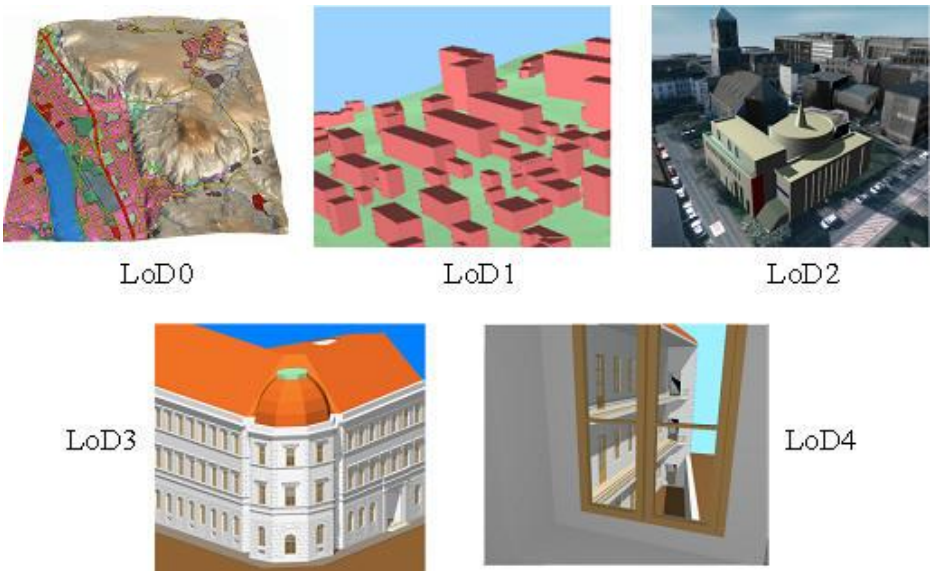

Abbildung 2: CityGML Detaillierungsstufen LOD0–LOD4 (Kolbe 2009)

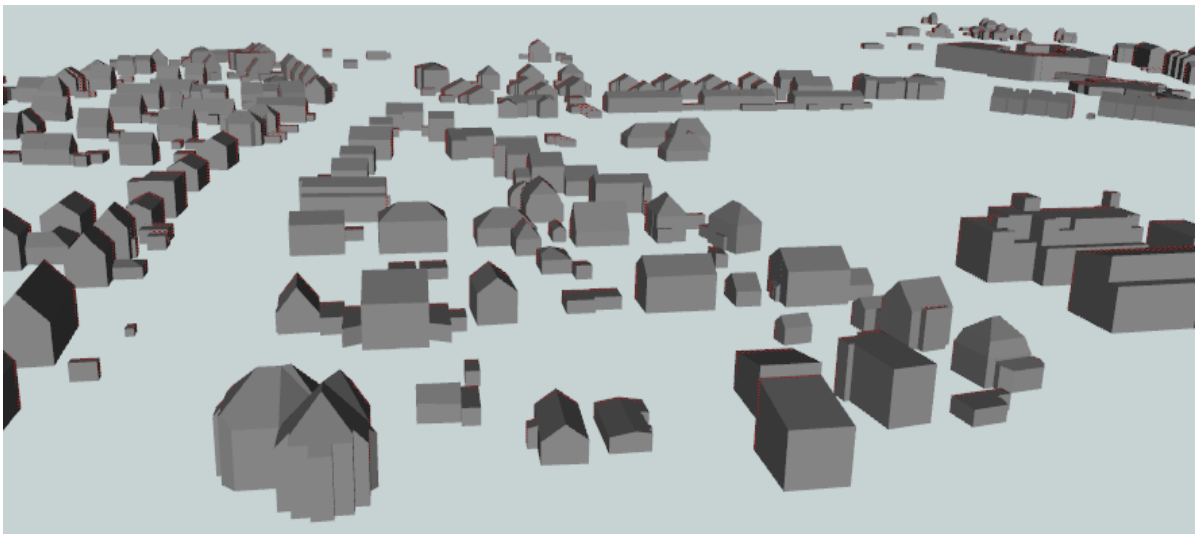

Abbildung 3: Ausschnitt aus dem CityGML-Beispieldatensatz Pullach i. Isartal (LDBV Bayern)

2.2.2 Konzept der CityGML-Module

Der CityGML-Standard wurde als GML 3.1.1 Anwendungsschema mit der ISO-Norm 19109 realisiert, welches neben Geometrierepräsentationen einige grundlegende Modellierungskonzepte bereitstellt. Durch den modularen Aufbau von CityGML sind unterschiedliche

fachliche Themen in jeweils eigene XML-Schemata mit separaten Namensräumen repräsentiert. Dadurch müssen Applikationen nicht das komplette CityGML-Schema integrieren, um CityGML-konform zu sein (Löwner et al. 2012). CityGML enthält alle Module, wie in Abbildung 4 dargestellt.

Alle CityGML-Module werden mit den in Klammern angegebenen Namensräumen kurz vorgestellt. Das Basismodul bildet das CityGML-Core-Modul. Alle weiteren Klassen bauen auf diesem Modul auf, weshalb es in jedem validen System vorhanden sein muss. Das Modul Relief (dem) repräsentiert die geometrische Struktur der Erdoberfläche. Das Modul Building (bldg) repräsentiert Gebäude und Gebäudekomponenten. Im Modul Bridge (brid) sind Brücken und deren Komponenten modelliert. Analog zu diesem Modul werden im Modul Tunnel (tun) Tunnel und deren Komponenten repräsentiert. Im Modul WaterBody (wtr) sind Wasserflächen bzw. -körper modelliert.

Das Modul Transportation (tran) ist eine Repräsentation vom Verkehrsraum und stellt Straßen oder Schienenwege dar. Das Modul Vegetation (veg) repräsentiert einzelne oder flächenhaft verteilte Vegetationsobjekte, während das Modul CityFurniture (frn) weitere Objekte einer Stadtlandschaft, wie Verkehrsampeln und Reklameschilder, welche nicht durch ein anderes thematisches Modul abgebildet werden können, darstellt. Im Modul LandUse (luse) wird die physikalische oder biologische Landbedeckung bzw. die sozio-ökonomische Landnutzung abgebildet. Das Modul CityObjectGroup (grp) verkörpert applikationsspezifische Gruppierungen beliebiger CityGML-Objekte (Löwner et al. 2012).

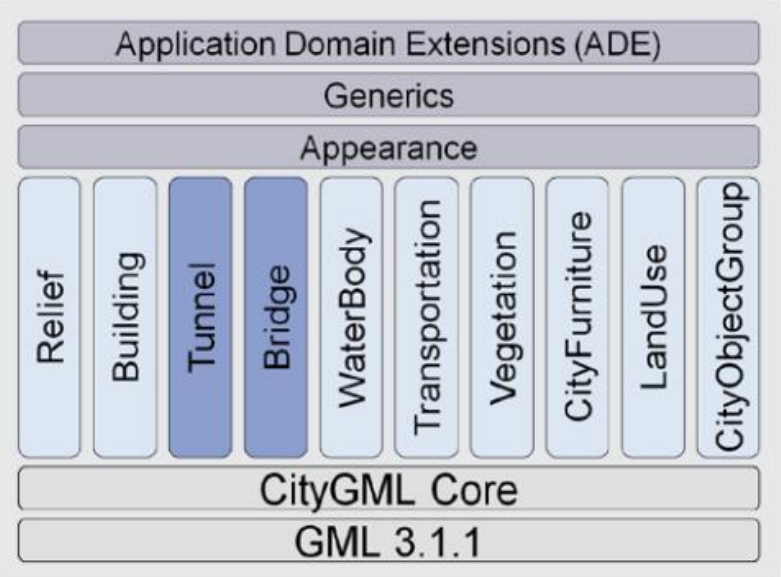

Abbildung 4: CityGML-Module (Kolbe 2009)

Durch die Modularisierung von CityGML sind alle Module, wenn diese einzeln vorkommen, in sich valide. Das heißt, dass diese eigenständige (räumliche) Objekte charakterisieren, die nicht voneinander abhängig sind. Diese vertikalen Module enthalten die semantische Modellierung für unterschiedliche Themenbereiche.

2.2.3 Semantik in 3D-Stadtmodellen

Semantik beschreibt die geometrischen Objekte eines CityGML-Modells näher. Mit Hilfe von Attributen und Beziehungen entsteht so ein komplexes Modell. Diese semantischen Informationen sind unerlässlich, um Analysen und komplexe Abfragen auf die Geometrien auszuführen. Das CityGML-Modell *Building* weist neben Geometrien Attributwerte auf, welche zusätzliche Informationen zu den Gebäuden beinhalten. Die Attribute beschreiben diese Objekte so deutlich, dass eine realistischere Darstellung möglich ist. Neben den Geometrieobjekten liefern die Attribute weitere Informationen zur Ausrichtung und Lage der Geometrie, wodurch umfangreiche thematische und räumliche als auch kombinierte Abfragen an ein CityGML-Modell gestellt werden können (Gröger et al. 2012).
Das Gebäude wird durch eine Vielzahl von Attributen beschrieben, die nach dem Level of Detail variieren. In Abbildung 5 ist der Zusammenhang aus Geometrie und Semantik veranschaulicht und demonstriert zugleich die Abhängigkeit voneinander.

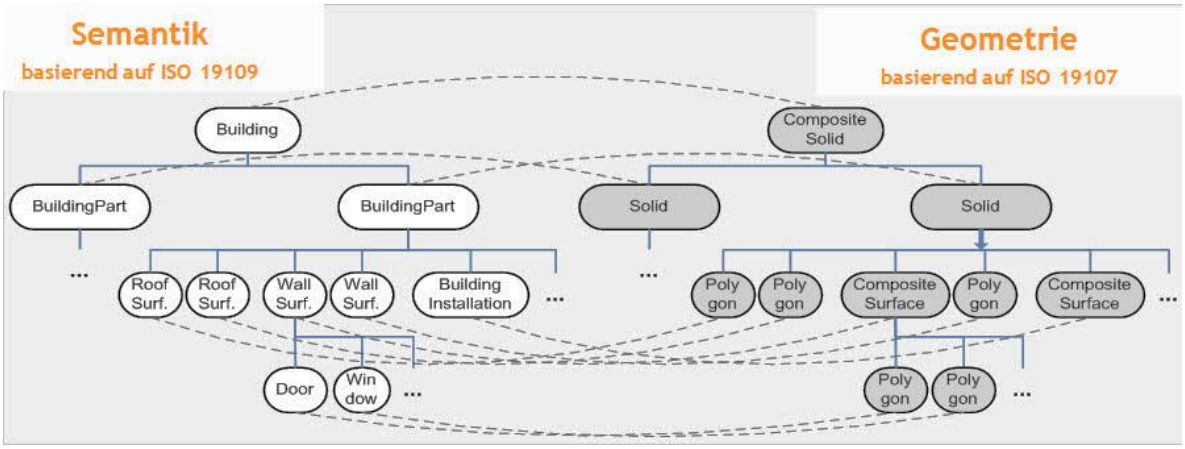

Abbildung 5: Kohärenz von Semantik und Geometrie (Nagel et al., 2008)

Das CityGML-Modell ist ein semantisches Modell und verwendet für die Objekte in 3D-Stadtmodellen (Gebäude, Brücken, Tunnel, etc.) Klassendefinitionen. Abbildung 6 veranschaulicht einen Teil dieses semantischen Modells für die Beschreibung von Gebäuden und basiert auf der ISO 19109. Diese Norm definiert Regeln für die Erstellung von Anwendungsschemata, wodurch diese auf eine einheitliche Weise erstellt werden können. Dies dient unter anderem der Generierung, Verarbeitung, Visualisierung und dem Transfer von Geodaten zwischen unterschiedlichen Anwendern und Systemen (Kutzner und Eisenhut 2010).

Das CityGML-Modell wurde als UML-Klassendiagramm modelliert. Die Basisklasse ist die abstrakte Klasse _CityObject. Diese erbt die Attribute name, description und gml:id von der GML-Superklasse _Feature. Die Attribute creationDate und terminationDate werden von _CityObject zur Verfügung gestellt und ermöglichen zeitabhängige Modellierungen von Objekten. Durch ExternalReference kann _CityObject mit externen Datensätzen oder Objekten verknüpft werden (Gröger et al. 2012). Die Aggregation von _CityObject nennt man CityModel, welche wiederum eine Subklasse der GML-Superklasse FeatureCollection ist, wie in Abbildung 10 zu sehen.

Das Building-Modul ist für die Thematik der vorliegenden Arbeit von zentraler Bedeutung. Die Pivotklasse des Gebäudemoduls ist die abstrakte Klasse _AbstractBuilding, von welcher sich die beiden nicht-abstrakten Klassen Building und BuildingPart ableiten und alle Attribute ihrer Superklasse erben, wie in Abbildung 7 veranschaulicht. Ein Gebäude kann aus einem Building und einem oder mehreren BuildingParts bestehen (Kolbe 2009).

Die zentrale abstrakte Klasse vom Modul Building ist _AbstractBuilding. Die semantischen Attribute yearOfConstruction, yearOfDemolition, storeysAboveGround, storeysBelowGround, storeyHeightsAboveGround und storeyHeightsBelowGround sind darin enthalten. Des Weiteren sind die bereits erwähnten Attribute class, function und usage enthalten. Das Attribut class enthält eine Klassifikation des Gebäudes nach privater oder gewerblicher Nutzung. In den Attributen function und usage sind Informationen über die geplante bzw. tatsächliche Nutzung des Gebäudes hinterlegt (Löwner et al. 2012).

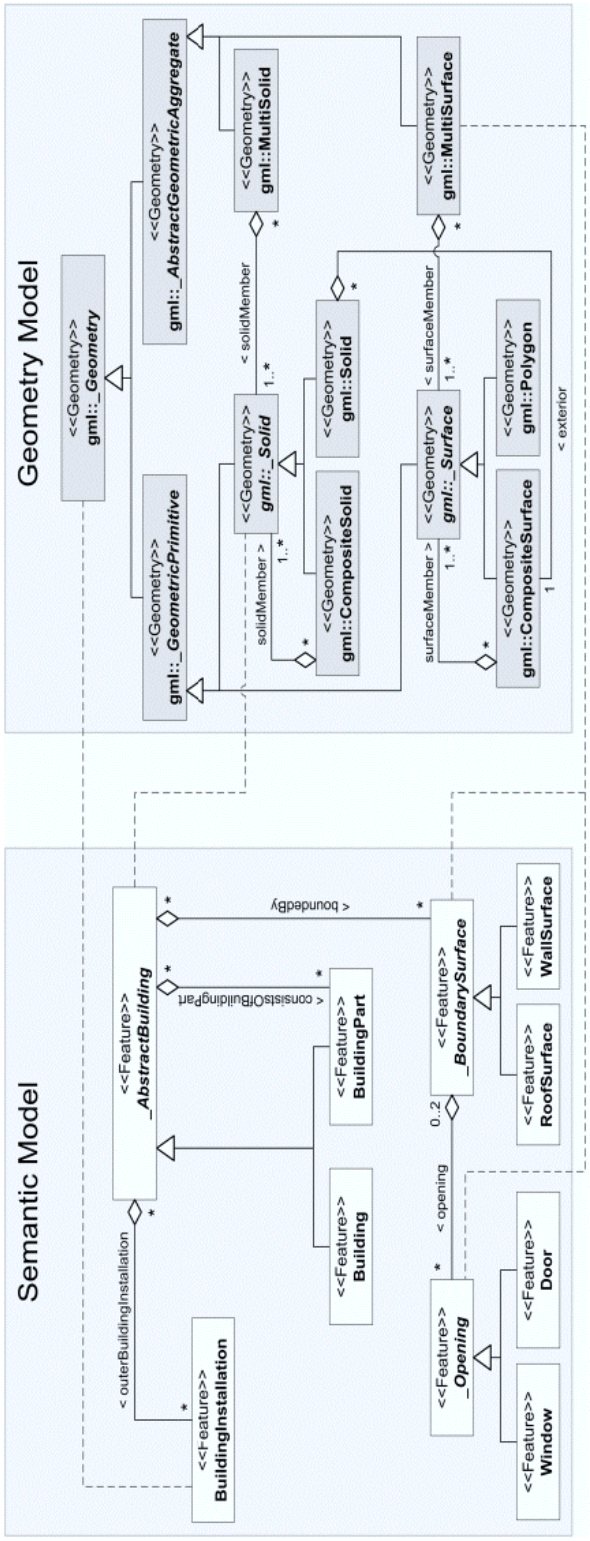

Abbildung 6: Gegenüberstellung der UML-Klassendiagramme des semantischen und geometrischen Modells von CityGML (links: Ausschnitt des Gebäudemodells - ISO 19109, rechts: Ausschnitt aus dem ‘Spatial Schema’ ISO 19107) (Stadler, A., & Kolbe, T. H. 2007)

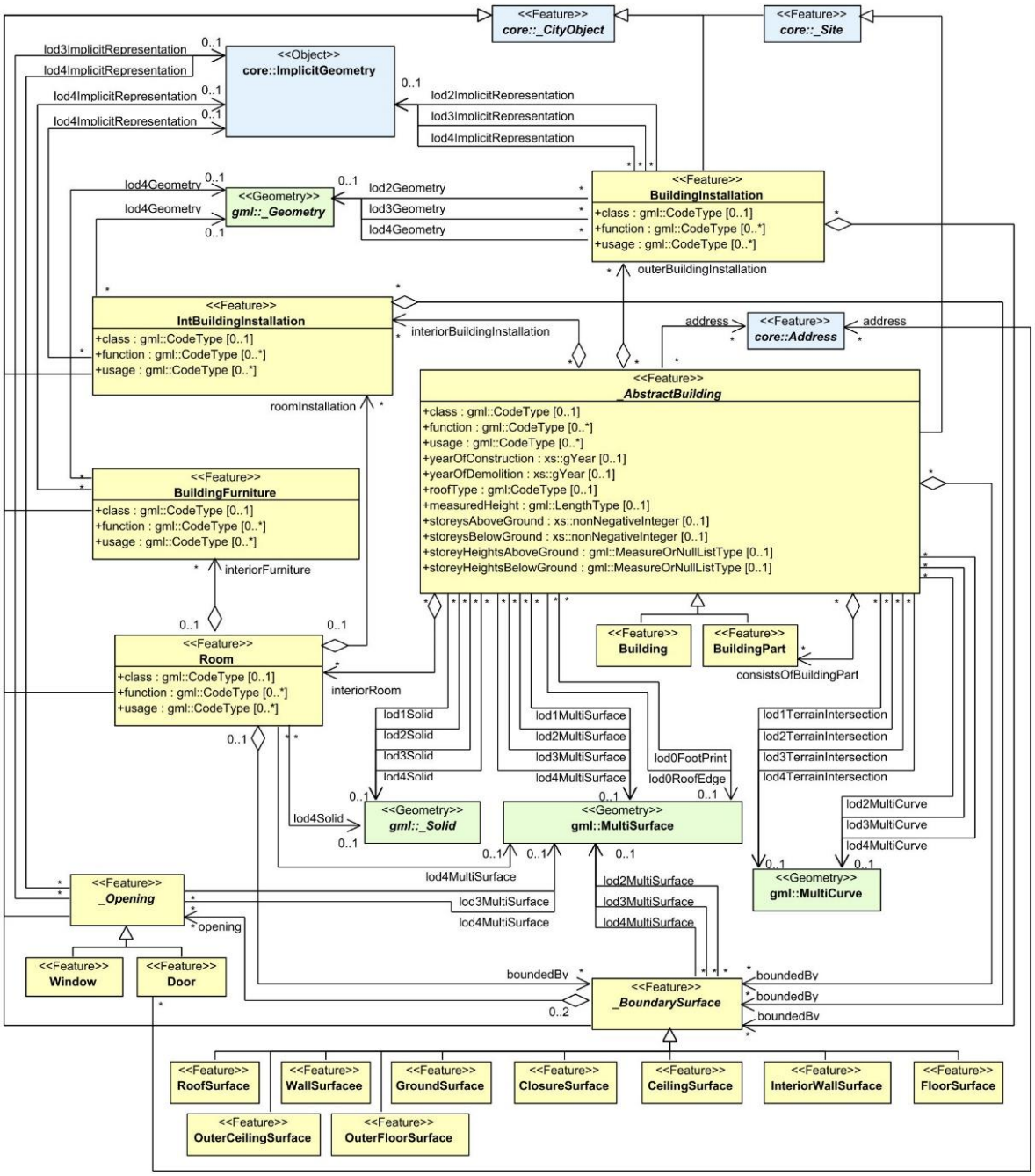

Abbildung 7: CityGML Building-Modul (Gröger et al. 2012)

2.2.4 Erweiterungsmöglichkeiten des CityGML-Modells

CityGML-Daten können spezifisch erweitert werden. Dafür gibt es zwei Optionen. Die erste Option ist die sogenannte Application Domain Extension (ADE), welche in einer XML-

Schema-Definitionsdatei mit einem eigenen Namensraum definiert wird. Dadurch ist die Erweiterung formal festgelegt. Für die Validierung des erweiterten CityGML-Modells ist diese Schemadatei notwendig. Beispiele hierfür sind die CityGML Energy ADE oder die CityGML Noise ADE. Innerhalb eines CityGML-Modells können auch mehrere ADEs verwendet werden (Kolbe 2009).

Die zweite Option ist das Verwenden von generischen Objekten (*GenericCityObjects*) oder generischen Attributen (*_genericAttributes*), welche innerhalb des Moduls *Generics* definiert sind. Ein *_CityObject* kann beliebig viele *_genericAttributes* enthalten, für die allerdings jeweils der Name, der Typ und der Wert im CityGML-Datensatz gegeben sein müssen (Kolbe 2009).

2.3 Das AdV-Profil für CityGML

Die Vermessungsverwaltungen der Länder, kurz AdV, haben sich für das Liegenschaftskataster als optimale Produktionsgrundlage für die Erstellung von 3D-Gebäudemodellen entschieden. Man hat für die Erstellung der 3D-Gebäudemodelle den CityGML-Standard und die Reduktion auf das CityGML 1.0-Schema festgelegt. Der Aufbau von Gebäudemodellen soll deutschlandweit in einer ersten Detailstufe zunächst als sogenanntes Klötzchenmodell im LOD1 erfolgen. Das Gebäudemodell im LOD1 sollte ab dem Jahr 2013 bundesweit zur Verfügung stehen, wobei die Erfassung zunächst als zweidimensionales Gebäude mit den dazugehörigen dreidimensionalen Informationen erfolgte. Die Bereitstellung der Gebäudemodelle im LOD2 mit standardisierten Dachformen steht noch aus, jedoch haben einige Bundesländer bereits mit der Erstellung begonnen (AdV 2015a).

Die Zentrale Stelle Hauskoordinaten und Hausumringe, kurz ZSHH, übernimmt die Bereitstellung der CityGML-Daten. Das bestehende AAA-Modell der GeoInfoDok wurde um die 3D-Gebäudemodelle erweitert, wodurch man dem Ziel einer bundeseinheitlichen Bereitstellung von 3D-Gebäudemodellen in den beiden Detaillierungsstufen näher kommt. Mit der Einführung der GeoInfoDok 7.0 sollen die Gebäudemodelle zukünftig als vollständige 3D-Volumenkörper vorliegen (AdV 2015a).

Inhaltlich unterscheidet sich das AdV-CityGML-Profil vom ursprünglichen CityGML-Modell. Diesem wurden Qualitätsangaben hinzugefügt, um die Stadtmodelle inhaltlich anzureichern. Ein Teil der bereits existenten Qualitätsangaben wurde um spezifische Attributwerte erweitert.

Diese Anreicherung erfolgt durch Attribute, die diese Stadtmodelle näher beschreiben. So werden z.B. Höhenangaben aus unterschiedlichen Erfassungsmethoden abgeleitet. Diese Attribute weisen einen Wert auf, der mit Hilfe einer Codeliste wiederum dieser Erfassungsmethode zugeordnet werden kann.

Die Attribute, die über Codelisten beschrieben werden, werden über integer-Werte (ganzzahlige Werte) eingeschränkt.

Im AdV-CityGML-Profil werden darüber hinaus weitere Attributdaten erfasst. Insgesamt sind folgende Attribute als Pflichtattribute vorhanden: Höhe des Gebäudes; Objektidentifikator; Referenz auf das ALKIS/ALK-Gebäude; Ableitungsdatum; Kennung und Gebäude- bzw. Bauwerksfunktion; Datenquelle Dachhöhe; Datenquelle Lage; Datenquelle Bodenhöhe und Gemeindeschlüssel (8-stellig mit vorangestelltem 2-stelligen Bundeslandkürzel). Hierzu gehören auch optionale Attribute, die geprüft werden müssen. Diese beinhalten roofType, minimale Gebäudehöhe (kein Pflichtattribut des LOD2) und maximale Gebäudehöhe (kein Attribut des LOD2) (AdV 2015d).

Bei den generischen Attributen wird der Eintrag über die Codelisten abgeglichen. Datentypen werden hier nicht gesondert definiert. Die Datentypen aller übrigen Attribute sind in der Profil-xsd definiert. Die AdV Schemadateien sind unter: http://repository.gdi-de.org/schemas/adv/citygml/ abgelegt. Dazu gehören: *Appearance*, *Building*, *Cityobjectgroup* und *Generics*.

Generische (benutzerdefinierte) Attribute können verwendet werden, um Attribute, die nicht ausdrücklich von dem CityGML-Schema abgedeckt werden, zu vertreten. Generische Attribute müssen mit Vorsicht verwendet werden. Diese werden nur dann verwendet, wenn es keine sachgemäßen Attribute im CityGML-Schema gibt. Andernfalls können Probleme hinsichtlich der semantischen Interoperabilität auftreten. Ein generisches Attribut hat einen Namen und einen Wert. Dieser Wert kann als IntAttribute, StringAttribute, etc. beschrieben sein. Laut AdV dürfen die Inhalte des Datentyps *genericAttribute* nur vom Typ *string* sein

(AdV 2015a). In Tabelle 1 sind die definierten generischen Attribute und deren entsprechende Bezeichnung im AdV-CityGML-Modell abgebildet.

Pflichtattribute laut AdV-CityGML-Profil	Attributname in CityGML
Höhe des Gebäudes aus der Differenz der Dachhöhe und der Bodenhöhe	measuredHeight
Objektidentifikator	gml:id
Referenz auf das ALKIS/ALK-Gebäude	externalReference
Ableitungsdatum	creationDate
Kennung und Gebäude- bzw. Bauwerksfunktion	function
Datenquelle Dachhöhe	stringAttribute name="Datenquelle-Dachhoehe"
Datenquelle Lage	stringAttribute name="Datenquelle-Lage
Datenquelle Bodenhöhe	stringAttribute name="Datenquelle-Bodenhoehe
Gemeindeschlüssel	stringAttribute name=" Gemeindeschluessel"

Tabelle 1: AdV-Pflichtattribute und Attributname in CityGML

2.3.1. Module des AdV-CityGML-Profils

Der modulare Aufbau des CityGML-Datenmodells wurde bereits im vorherigen Kapitel erwähnt. Das AdV-CityGML-Profil ist in Anlehnung an das CityGML-Modell entstanden und ergibt sich als Reduktion des CityGML 1.0-Schemas (AdV 2015a). Die Beschreibung des CityGML-Modells basiert auf der OGC-Spezifikation CityGML Version 1.0.0, OpenGIS City Geography Markup Language (CityGML) Encoding Standard 08-007r1 (AdV 2015b). Hier soll nun auf die relevanten Module eingegangen und die Unterschiede gegenüber dem CityGML 1.0-Schema aufgezeigt werden. Das AdV-CityGML-Profil für den Detaillierungsgrad LOD1 unterscheidet sich vom gewöhnlichen CityGML-Modell. In Abbildung 8 und Abbildung 9 sind die CityGML-Profile für LOD1 und LOD2 der AdV abgebildet.

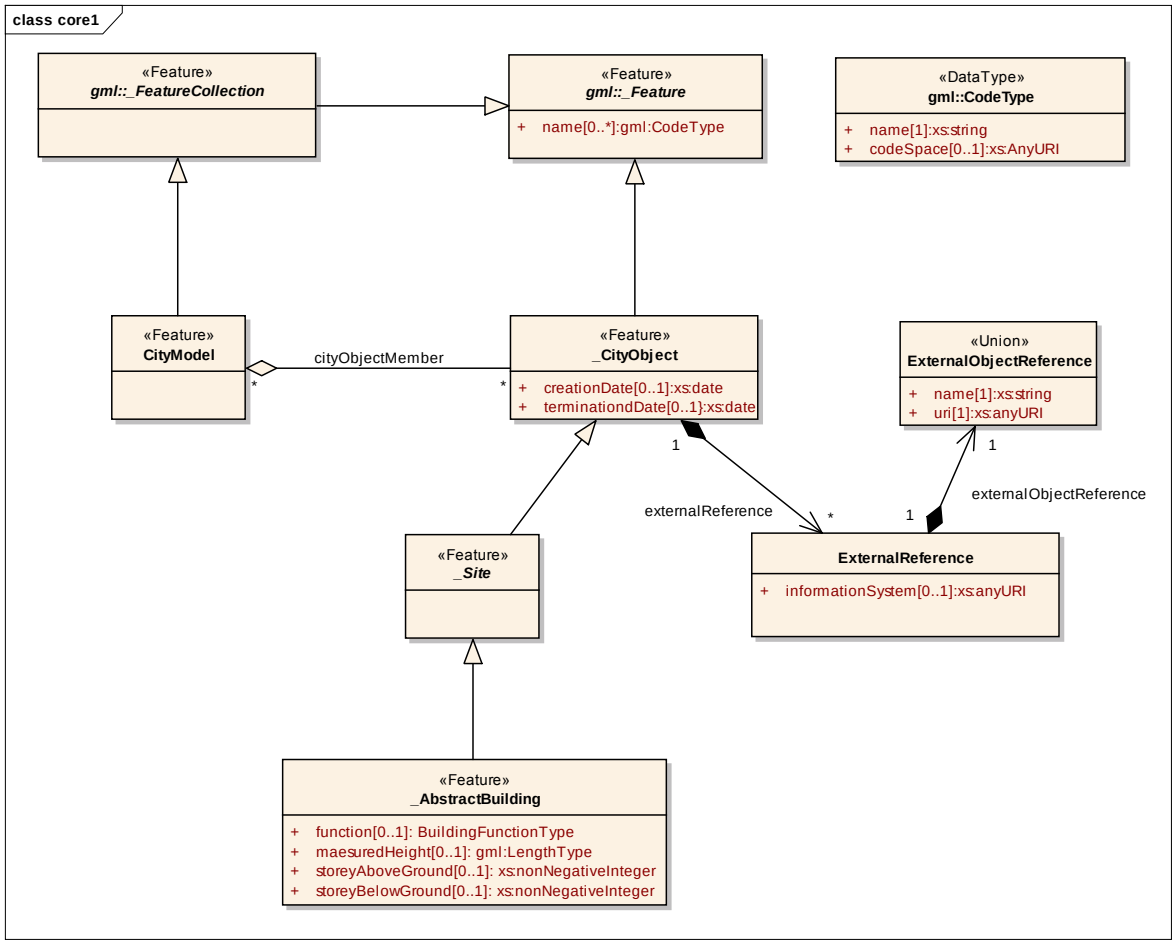

Abbildung 8: CityGML-Profil für LOD1 der AdV (AdV 2015a)

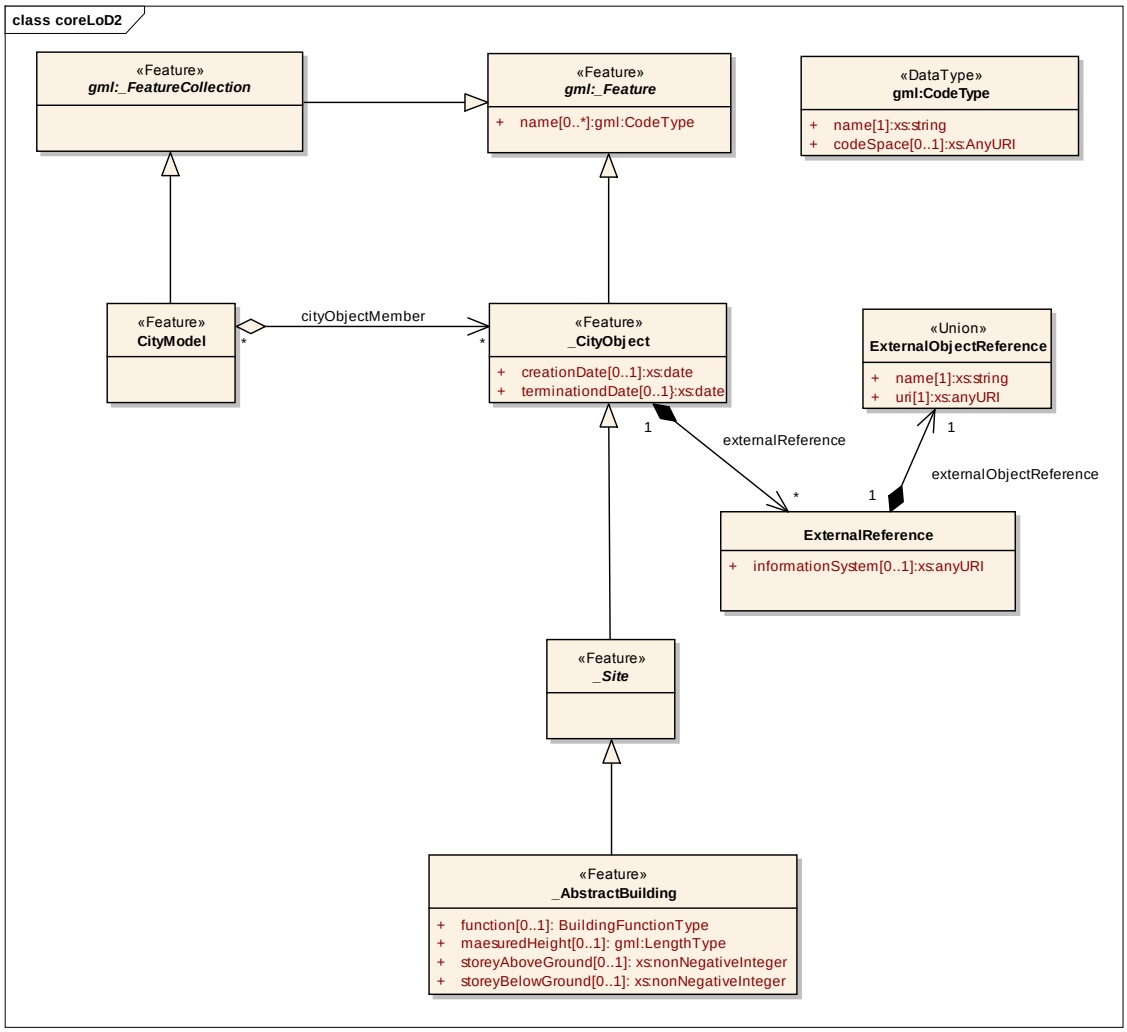

Abbildung 9: CityGML-Profil für LOD2 der AdV (AdV 2015a)

2.3.1.1 Modul Core

Das Modul *Core*, dargestellt in Abbildung 10, weist im Detaillierungsgrad LOD1 Änderungen gegenüber dem CityGML 1.0-Schema auf. Für das AdV-CityGML-Profil wurde die Generalisierungsrelation entfernt. Für den Detaillierungsgrad LOD2 trifft dies ebenfalls zu (AdV 2015a). Das *Core*-Modul von CityGML beinhaltet die elementaren Bestandteile des CityGML-Datenmodells und weist abstrakte als auch nicht-abstrakte Klassen auf. Diese nicht-abstrakten Klassen beschreiben thematische Klassen, die von mehr als einem Erweiterungsmodul verwendet werden können (Gröger et al. 2012).

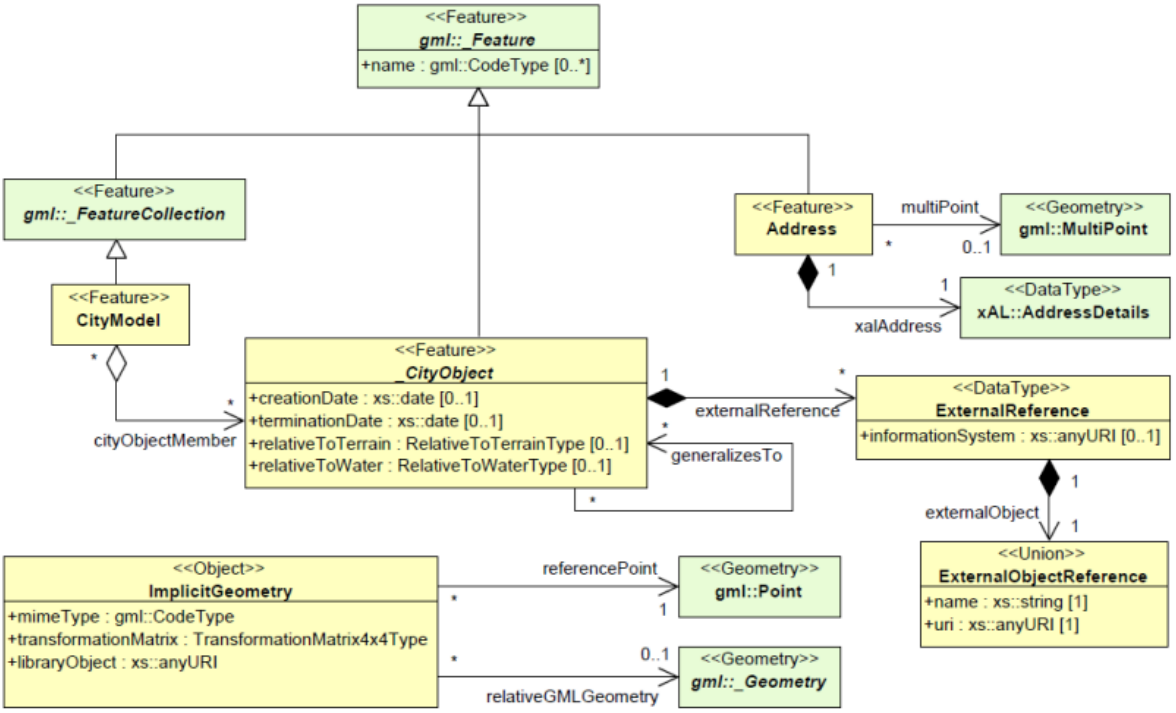

Abbildung 10: Ausschnitt des CityGML-UML-Diagramms des Core Moduls (Gröger et al. 2012)

2.3.1.2 Modul Building

Das thematische Modul *Building* ist das Basismodul innerhalb von CityGML, welches in dieser Arbeit von zentraler Bedeutung ist. Es stellt die Repräsentation von Gebäuden und Gebäudekomponenten dar.

Im Vergleich zum Building-Modul des CityGML-Profils ist für das AdV-CityGML-Profil eine Reduzierung auf LOD1-Geometrien vorgenommen worden, wobei ausschließlich LOD1-Solid zugelassen ist. Die semantischen Klassen für Objekte (*wallsurface*, *roofsurface*, etc.) ab dem Detaillierungsgrad LOD2 wurden entfernt. Des Weiteren sind externe Gebäudeinstallationen nicht zugelassen. Die Attribute *function* (Gebäudefunktion) und *measuredHeight* (gemessene Höhe) sind abweichend vom Standard CityGML als Pflichtattribute modelliert worden. Die Gebäudefunktion setzt sich zusammen aus der Kennung der Objektart und der Gebäudenutzung. Es können aber weiterhin, wie in Abbildung 11 zu sehen, externe Referenzen an den Gebäudeobjekten angebracht werden (AdV 2015a).

Darüber hinaus wurden Änderungen auch für den LOD2 vorgenommen. Eine Reduzierung auf die LOD1- und LOD2-Geometrien, semantische Begrenzungsflächen (*BoundarySurface*) und Solid als Referenzen auf die Begrenzungsflächen zählen hierzu. Weiterhin sind Gebäudeteile und externe Gebäudeinstallationen explizit zugelassen.

Wie bei dem Detaillierungsgrad LOD1 sind die Attribute *functio*n (Gebäudefunktion) und *measuredHeight* (gemessene Höhe) abweichend vom Standard CityGML als Pflichtattribute modelliert. Es können externe Referenzen an den Gebäudeobjekten angebracht werden. Im Detaillierungsgrad LOD2 ist die Nutzung der Klassen *RoofSurface, WallSurface, GroundSurface* und *ClosureSurface* obligatorisch (AdV 2015a). Dies ist in Abbildung 12 dargestellt.

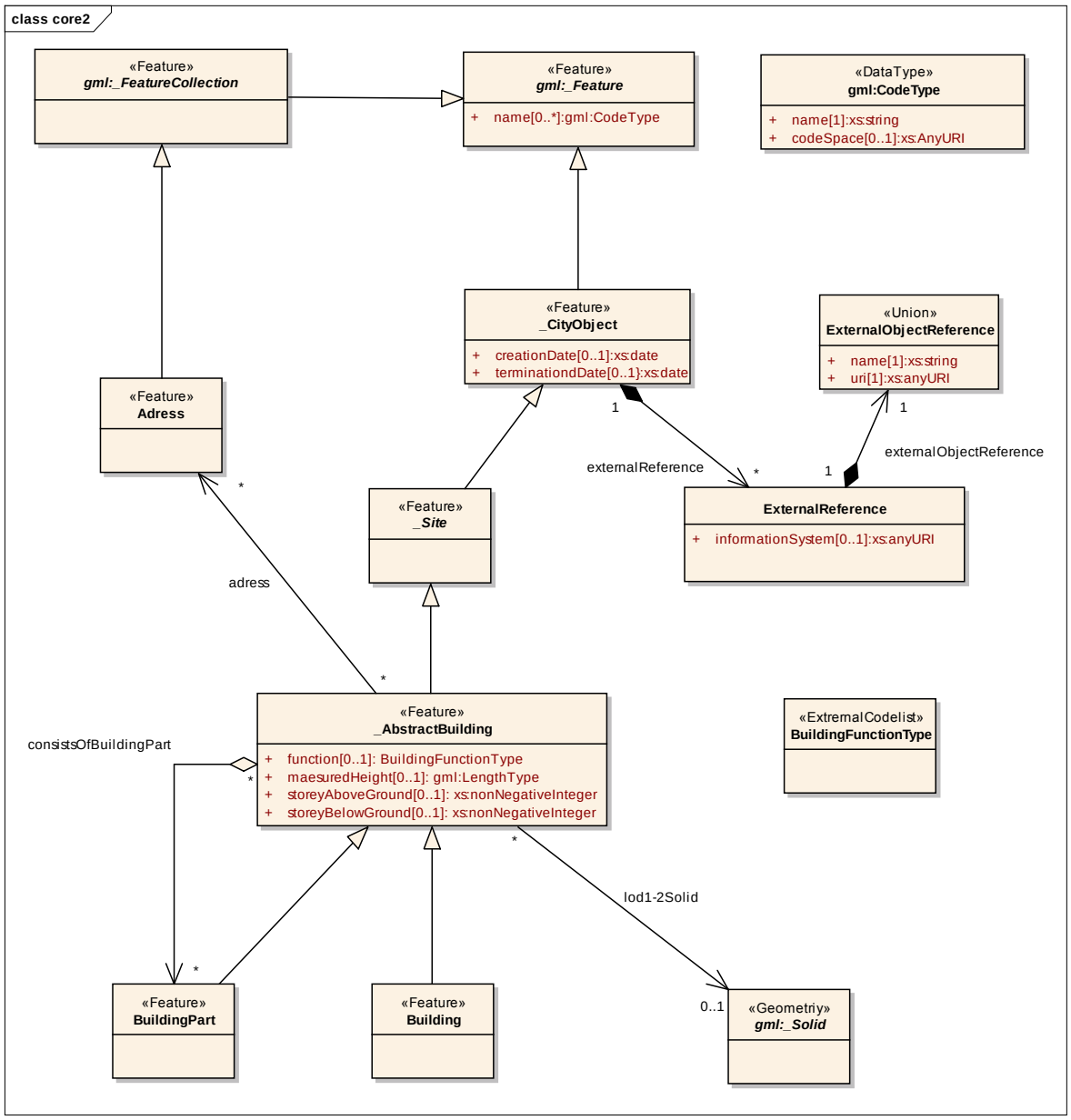

Abbildung 11: CityGML-Profil für LOD 1 der AdV (Building) (AdV 2015a)

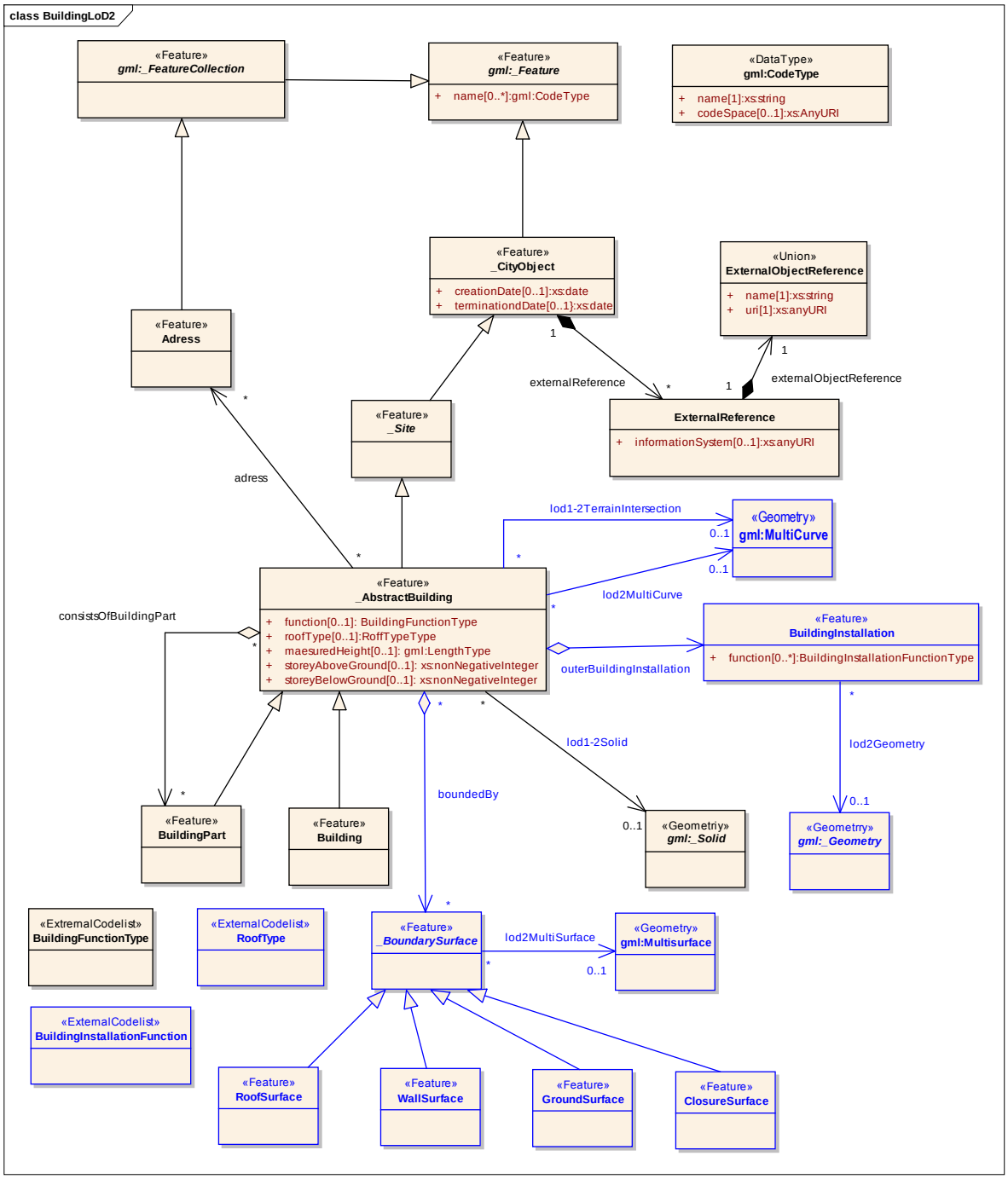

Abbildung 12: CityGML-Profil für LOD 2 der AdV (Building) (AdV 2015a)

2.3.1.3 Modul Generics

Im vorherigen Abschnitt wurde erwähnt, dass CityGML-Daten spezifisch erweitert werden können. Für das AdV-CityGML-Profil wurde zwischen der Abbildung in CityGML nach den Qualitätsmerkmalen ISO 19115 und 19139 und der generischen Variante (genericAttributes) entschieden. Das Ergebnis fiel zugunsten einer einfacheren Nutzung und Lesbarkeit der generischen Modellierung der Metadaten aus. Des Weiteren dürfen die Inhalte im Datentyp genericAttribute nur vom Typ "string" (Abbildung 12) sein und sollen nur die Inhalte der Codelists der Qualitätsangaben enthalten (AdV 2015a).

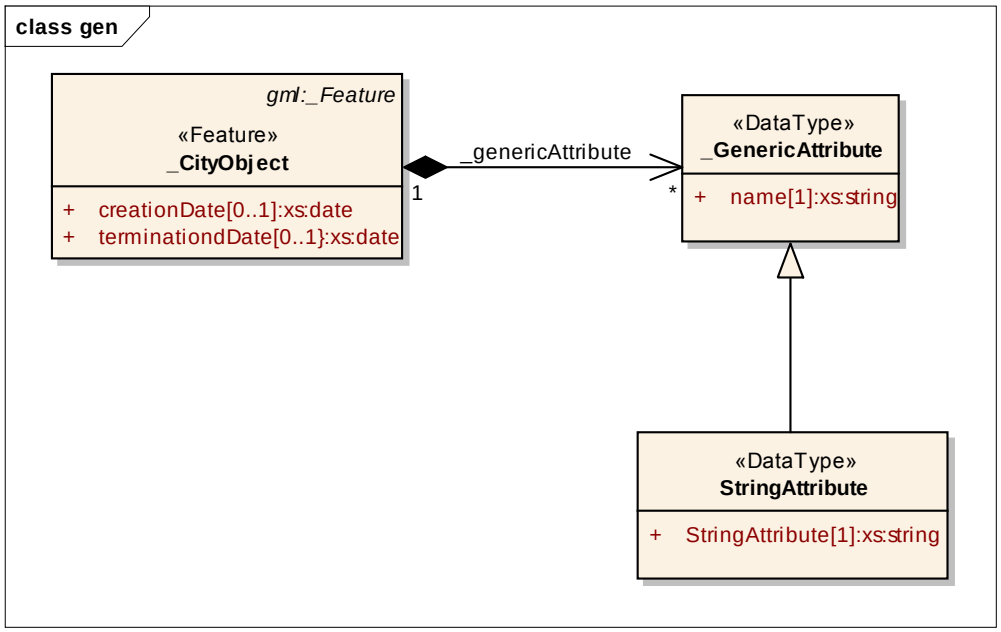

Abbildung 13: CityGML-Profil für LOD1/LOD2 der AdV (Generic) (AdV 2015a)

Für das AdV-CityGML-Profil LOD1 wurden sämtliche anderen Module entfernt. Dazu gehören: Appearance (app), CityFurniture (frn), CityObjectGroup (grp), LandUse (luse), Relief (dem), Transportation (tran), Vegetation (veg), WaterBody (wtr) und TexturedSurface (tex). Darüber hinaus wurden folgende externe Module in LOD1 unverändert übernommen: Geo-

graphy Markup Language (gml), Extensible Adress Language (xAL) und Schematron Assertion Language (sch). Für das LOD2 ist neben den zuvor genannten Modulen des LOD1 auch das Modul Appearance im AdV-CityGML-Profil erhalten geblieben (AdV 2015a).

2.3.2 Qualitätsangaben und generische Attribute

Die Qualitätsangaben mit Hilfe der Codelisten ermöglichen es, bereits existierende Attribute in CityGML‑Klassen semantisch anzupassen bzw. diese zu erweitern. Eine Erweiterung einer existierenden Klasse um zusätzliche Attribute ist nicht möglich. Ebenso wenig sind vollständig neue Klassen zu definieren (Löwner et al. 2012). Wie bereits im vorherigen Kapitel angesprochen, kann das Modul *Generics* hinsichtlich weiterer Attribute ergänzt werden. Diese generischen Attribute stellen für das CityGML-Profil eine Erweiterung dar, um das dieses hinsichtlich weiterer Anwendungsbereiche beliebig weiterentwickelt werden kann.
Die Anzahl dieser Attribute unterscheidet sich mit den Detaillierungsstufen. Die Tabelle 2 zeigt die Qualitätsangaben des AdV-CityGML-Profils für LOD1 und LOD2 und die Verteilung auf Building oder BuildingPart (wenn nicht als Solid modelliert). Die mit „-„ gekennzeichneten Attribute sind nicht zu führen, während „x“ Pflichtangaben und „o“ optionale Angaben sind. Wenn keine Gebäudeteile gebildet werden, sind die Attribute dem Building zugeordnet.

Attribut	Building	Buildingpart	Bemerkungen
Function	x	-	Es sind nur Werte nach AdV-Codelists zugelassen
RoofType	-	o	nur LOD2
MeasuredHeight	-	x	relative Höhe; keine zwingende Abhängigkeit zur Geometriehöhe
StoreysAboveGround	-	o	
StoreysBelowGround		o	
ExternalReference	x	-	ALKIS- bzw. ALK-Kennzeichen

Generics (AmtlicherGemeinde-schlüssel)	x	-	
Generics (Qualitätsangaben) Datenquelle Dachhöhe Datenquelle Lage Datenquelle Bodenhöhe Bezugspunkt Dach (nur LOD1)	-	x	
AddressFeature (Lagebezeich-nung)	o	-	
gml:name (Gebäudename)	o	-	Es sind nur Gebäudeeigennamen zugelassen
appearance	-	o	
TerrainIntersectionCurve (Gelän-deschnittlinie)	-	o	

Tabelle 2: Qualitätsangaben des AdV-CityGML-Profils (AdV 2015a)

Das Attribut „DatenquelleDachhoehe“ beschreibt das Verfahren zur Ermittlung der Höhe bei LOD1 oder LOD2-Volumenkörpern. „DatenquelleLage“ schildert das Verfahren und die Quelldaten für die lagemäßige Festlegung der LOD1 oder LOD2-Körper. Das generische Attribut „DatenquelleBodenhoehe“ wiederum charakterisiert das Verfahren und die zugrunde liegenden Daten zur Ermittlung der absoluten Bodenhöhe. Das Attribut „BezugspunktDachhoehe“, welches nur im LOD1 vorkommt, beschreibt den Bezugspunkt einer vom Flachdach abweichenden Dachform. Dass das AdV-CityGML-Profil beliebig erweitert werden kann, zeigen die CityGML-Datensätze des LDBV Bayern. Diese wurden um weitere Qualitätsangaben über das AdV-CityGML-Profil hinaus erweitert. Die zusätzlichen Attribute sind in der Tabelle 3 *kursiv* geschrieben.

Produktstandard	**CityGML**
Datum der Übernahme der Grundrisse aus dem Liegenschafts-kataster	StandLK
Gemeindeschlüssel	Gemeindeschluessel
Datenquelle Dachhöhe	DatenquelleDachhoehe
Datenquelle Lage	DatenquelleLage
Datenquelle Bodenhöhe	DatenquelleBodenhoehe
Bezugspunkt Dach	BezugspunktDachhoehe (nur LOD1)
Höhe des tiefsten Gebäudepunktes	Hoehe Grund
Höhe des höchsten Daches	Hoehe Dach
Höhe der niedrigsten Dachtraufe	NiedrigsteTraufeDesGebaeudes
Erzeugungsart der Dachformerkennung	Methode
Inhalt der Fläche	Flaeche
Dachneigung	Dachneigung
Dachorientierung	Dachorientierung
Höchster Punkt der Fläche (relativ)	Z_MAX
Höchster Punkt der Fläche (über NHN)	Z_MAX_ASL
Niedrigster Punkt der Fläche (relativ)	Z_MIN
Niedrigster Punkt der Fläche (über NHN)	Z_MIN_ASL

Tabelle 3: Qualitätsangaben des CityGML-Profils der AdV und zusätzliche des LDBV Bayern (*kursiv*) (Roschlaub, Robert, Dr. (LVG) 2014).

2.3.3 Codelisten der Qualitätsangaben

Ein Teil der Qualitätsangaben werden mit Hilfe von Codelisten abgebildet. Hierzu zählen: Datenquelle Dachhöhe, Datenquelle Lage, Datenquelle Bodenhöhe und Bezugspunkt Dach (AdV 2015a). Das wichtigste Konzept zur semantischen Anpassung und Erweiterung von CityGML sind Codelisten. Eine Codeliste beinhaltet zulässige Werte, die ein bestimmtes Attribut annehmen kann. Im Vergleich zum ebenfalls verwendeten Konzept der Enumerationen werden Codelisten nicht im XML‑Schema repräsentiert, sondern mit Hilfe von externen GML‑Dictionaries. Diese Codelisten definieren einerseits zulässige Werte und andererseits

weisen sie jedem Code einen eindeutigen Identifier zu, der die semantische Bedeutung des Codes verdeutlicht. In Tabelle 4 sind beispielhaft die Codelistenwerte und die dazugehörigen Bedeutungen für das Attribut *DatenquelleDachhoehe* veranschaulicht. Diese Codelisten können durch den Nutzer abgeändert oder erweitert werden. Es können sogar neue Listen generiert werden, um die Semantik der Attribute an seine Anforderungen anzupassen (Löwner et al. 2012).

Bedeutung	Wert
LASERSCAN	1000
STOCKWERKE	2000
STANDARD	3000
PHOTOGRAMMETRIE - MANUELL	4000
PHOTOGRAMMETRIE - AUTOMATISCH	5000
MANUELL	6000

Tabelle 4: Herstellungsprozess der Dachhöhenermittlung für das Attribut *DatenquelleDachhoehe* (AdV 2015a)

2.3.4 Prüfplan der relevanten Attribute

Die Anforderungen der AdV stellen die Attribute dar, welche innerhalb der CityGML-Daten vorhanden sein müssen. Neben der Existenz der Daten müssen diese auch valide Attributwerte beinhalten. Für die Prüfung stellen somit die Attribute als auch gültige Attributwerte einen Prüfplan dar, auf der die zu prüfenden Attribute zusammengefasst werden. Diese Attribute werden im Prüfprozess geprüft und in einem Prüfprotokoll ausgegeben. Für diese Arbeit werden aus dem Prüfplan die Prüfnummern 2100 bis 2630 versuchsweise umgesetzt. Diese gehören zur Testkategorie Profilkonformität. Die Prüfnummern dieser Testkategorien beziehen sich explizit auf die Attribute, welche in dieser Arbeit geprüft werden sollen, weshalb die restlichen Testkategorien in dieser Arbeit nicht behandelt werden.

2.3.4.1 Schemakonformität

Als Schemaprüfung wird die Prüfung auf Konformität gegen das xsd-Schema bezeichnet. Dabei geht es bei den CityGML LOD1- und LOD2-Datensätzen um die Schemaeinhaltung, also die Validierung gegen das xsd-Schema. Die Schemaprüfung ist ebenfalls ein Teil des Prüfplans, wird jedoch nicht in dieser Arbeit untersucht. In Abbildung 14 ist beispielhaft die Schemadatei für die Detaillierungsstufe LOD2 des Building Moduls dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- CityGML Profil der AdV, LoD 2, Version 11.10.2014-->
<!-- Editor: Ulrich Gruber, Kreis Recklinghausen, Kurt-Schumacher-Allee
1,  45657 Recklinghausen -->
<!-- Weitere Informationen unter www.adv-online.de -->
<xs:schema xmlns="http://www.opengis.net/citygml/building/1.0"
    xmlns:core="http://www.opengis.net/citygml/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.o
      pengis.net/gml"
    targetNamespace="http://www.opengis.net/citygml/building/1.0" ele
      mentFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:import namespace="http://www.opengis.net/gml" schemaLoca
      tion="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" />
    <xs:import namespace="http://www.opengis.net/citygml/1.0" schemaLoca
      tion="http://repository.gdi-de.org/schemas/adv/citygml/1.0/cityGML
      BaseLoD2.xsd"/>
    <!-- ========================================================== -->
    <!-- ===============CityGML Building module ==================== -->
    <!-- ========================================================== -->
    <xs:complexType name="AbstractBuildingType" abstract="true">
        <xs:complexContent>
            <xs:extension base="core:AbstractSiteType">
                <xs:sequence>
                    <xs:element name="function" type="BuildingFunc
                      tionType" minOccurs="0"/>
                     <xs:element name="roofType" type="RoofTypeType"
                      minOccurs="0"/>
                     <xs:element name="measuredHeight"
                      type="gml:LengthType" minOccurs="0"/>
                     <xs:element name="storeysAboveGround"
                      type="xs:nonNegativeInteger" minOccurs="0"/>
                     <xs:element name="storeysBelowGround"
                      type="xs:nonNegativeInteger" minOccurs="0"/>
                     <xs:element name="lod2Solid"
                      type="gml:SolidPropertyType" minOccurs="0"/>
                     <xs:element name="lod2TerrainIntersection"
                      type="gml:MultiCurvePropertyType" minOc
                        curs="0"/>
                     <xs:element name="outerBuildingInstallation"
                      type="BuildingInstallationPropertyType"
```

```
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="boundedBy" type="Bounda
                    rySurfacePropertyType" minOccurs="0" max
                    Occurs="unbounded"/>
                <xs:element name="consistsOfBuildingPart"
                    type="BuildingPartPropertyType" minOc
                    curs="0" maxOccurs="unbounded"/>
                <xs:element name="address" type="core:Address
                    PropertyType" minOccurs="0" maxOccurs="un
                    bounded"/>
                <xs:element ref="_GenericApplicationProper
                    tyOfAbstractBuilding" minOccurs="0" max
                    Occurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ========================================================== -->
<xs:element name="_AbstractBuilding" type="AbstractBuildingType" ab
   stract="true" substitutionGroup="core:_Site"/>
<!-- ========================================================== -->
<xs:element name="_GenericApplicationPropertyOfAbstractBuilding"
    type="xs:anyType" abstract="true"/>
<!-- ========================================================== -->
<xs:simpleType name="BuildingFunctionType">
```

Abbildung 14: Building Schemadatei LOD2 (“buildingLOD2.xsd “) (AdV)

Die Schemadateien sind online unter der Adresse: http://repository.gdi-de.org/schemas/adv/citygml/ zu finden. Es sind Schemadateien für das CityGML *Core* Modul (LOD1 und LOD2), CityGML *Appearence* Modul (LOD2), das CityGML *Building* Modul (LOD1 und LOD2), das CityGML *CityObjectGroup* Modul (LOD2), sowie Schemadateien für die *Generic Attributes* (LOD1 und LOD2) vorhanden.

Die manuelle Validierung von XML-basierten Dokumenten kann mit dem Programm XMLSpy der Firma Altova vorgenommen werden.

2.3.4.2 Profilkonformität

Analog zur Schemakonformität, welche eine Mindestanforderung an die CityGML-Datensätze darstellt, müssen die CityGML-Daten auch hinsichtlich des AdV-Profils validiert werden. Die Konformitätsbedingungen der CityGML-Datensätze zu dem AdV-Profil berücksichtigt neben der Schema-Validierung durch die Schema-XSD den Produktstandard, die

technischen Regelwerke und die Anforderungen der ZSHH (AdV 2015d). Mit Hilfe der allgemeingültigen, maschinenlesbaren Sprache Object Constraint Language, auf die am Anfang dieses Kapitels eingegangen wurde, ist eine Formulierung zur Validierung von Qualitätsangaben in Form von Attributdaten in CityGML-Modellen möglich. Das AAA-Datenmodell ist ein Beispiel für die Umsetzung von Zusicherungen. Darin wurde teilweise die automatische Überprüfung der Einhaltung der im AAA-Datenmodell beschriebenen Einschränkungen durch Schematron, eine XML-basierte, skalierbare Skriptsprache realisiert (Löwner et al. 2013). Dies soll im Rahmen dieser Arbeit für das AdV-CityGML-Profil erarbeitet werden.

3 Softwarewerkzeuge für die Datenprüfung

Als Entwicklungsumgebung für die Modellebene wird die Software Enterprise Architect 12 von Sparx Systems verwendet.
Für die Programmoberfläche wird die Skriptsprache Python in der Version 2.7 mit den Modulen *Tkinter* für die grafische Oberfläche, *lxml* für das parsen der XMI aus Enterprise Architect und FME von Safe Software für die Datenprüfung verwendet.
Für die Ausführung der Prüfung wird Safe Software FME 2015 - Build 15485 verwendet.
Im Laufe der Evaluation der Umsetzung einer Prüfsoftware wurden weitere Softwarekomponenten geprüft, aber schlussendlich nicht verwendet. Hier ist vor allem HALE – HUMBOLDT Alignment Editor zu nennen. Zur Arbeit auf der Datenebene fiel letztendlich die Entscheidung auf die Feature Manipulation Engine der Firma Safe Software aufgrund der Überführbarkeit von Teilen der OCL-Konstrukte in die interne Sprache von FME.

Für die Modellierung der OCL-Konstrukte wurde ursprünglich der Weg verfolgt, dies über die Software ShapeChange in Zusammenarbeit mit Enterprise Architect vorzunehmen. ShapeChange ist eine Open-Source-Software basierend auf Java von interactive instruments. Die Anwendung verarbeitet UML-Modelle für geografische Informationen gemäß der Standards von ISO/TC 211 und OGC. Es bietet direkte Unterstützung für UML-Modelle modelliert mit Enterprise Architect von Sparx Systems. Der Realisierung der Prüfsoftware wurde letztlich

der im nachfolgenden beschriebene Weg über die Programmiersprache Python der Vorrang gegeben, da hierdurch ein Großteil des Konvertierungsprozesses realisiert werden konnte.
Die am meisten verwendete Zielrepräsentierung ist XML Schema. Zusätzlich können auch Schematron-Dokumente aus den OCL-Constraints in den Anwendungsschemata abgeleitet werden.
Schematron ist eine Schemasprache zur Validierung von Inhalt und Struktur von XML-Dokumenten. Die Implementierung der Sprache ist über XSL-Transformationen realisiert, bedarf also keiner speziellen Implementierung, wie es bei den meisten anderen Dokumentstruktur-Definitionssprachen der Fall ist. XSL Transformation, kurz XSLT, ist eine Programmiersprache zur Transformation von XML-Dokumenten. Sie ist Teil der Extensible Stylesheet Language (XSL) und stellt eine turing-vollständige Sprache dar. XSLT baut auf der logischen Baumstruktur eines XML-Dokumentes auf und dient zur Definition von Umwandlungsregeln. XSLT-Programme, sogenannte XSLT-Stylesheets, sind dabei selbst nach den Regeln des XML-Standards aufgebaut (Jacinto et al. 2002; Tidwell 2008).
Schematron wurde speziell für Zusicherungen (assertions) erarbeitet und verwendet XPath zur Formulierung der Bedingungen (Becker 2004). Somit kann OCL vollständig implementiert und prozessiert werden. Dies wurde bereits in einer anderen Arbeit mit DresdenOCL in Anwendungen außerhalb des Geobereiches umgesetzt (Hussmann et al. 2002). HALE und DresdenOCL sind für diese Arbeit aber nicht weiter relevant und sollten nur als ein alternativer Lösungswegs innerhalb der verwandten Problematik aufgezeigt werden.

3.1 Modellierungssoftware Enterprise Architect

Enterprise Architect (EA) von der Firma SparxSystems ist ein Modellierungswerkzeug, dessen Kernfähigkeit die Erstellung von Softwaresystemen, sowie von Datenmodellen darstellt. Datenmodelle werden UML-konform nach der neusten UML-Spezifikation erstellt (Steinpichler und Kargl 2012).

3.1.1 Programmoberfläche

Ein Enterprise-Architect-Projekt besteht aus Modellen (models), Paketen (packages) und Diagrammen (diagrams). Im Projekt-Browser, auf der rechten Seite in Abbildung 13 werden die Modelle, Pakete und Diagramme hierarchisch strukturiert hinterlegt. In der Mitte der Abbildung 15 befindet sich die Diagrammansicht. Darin sind Bestandteile des gesamten Modells sichtbar. Diese lassen sich mit der Toolbox, auf der linken Seite in Abbildung 15 bearbeiten und zum Beispiel um Pakete oder Klassen erweitern. Diese lassen sich per Drag & Drop in das Modell im Projekt-Browser einfügen. Es können dabei Elemente umbenannt und Methoden oder Attribute gelöscht werden. Die Änderungen am Modell werden sofort in der Diagrammansicht angezeigt.

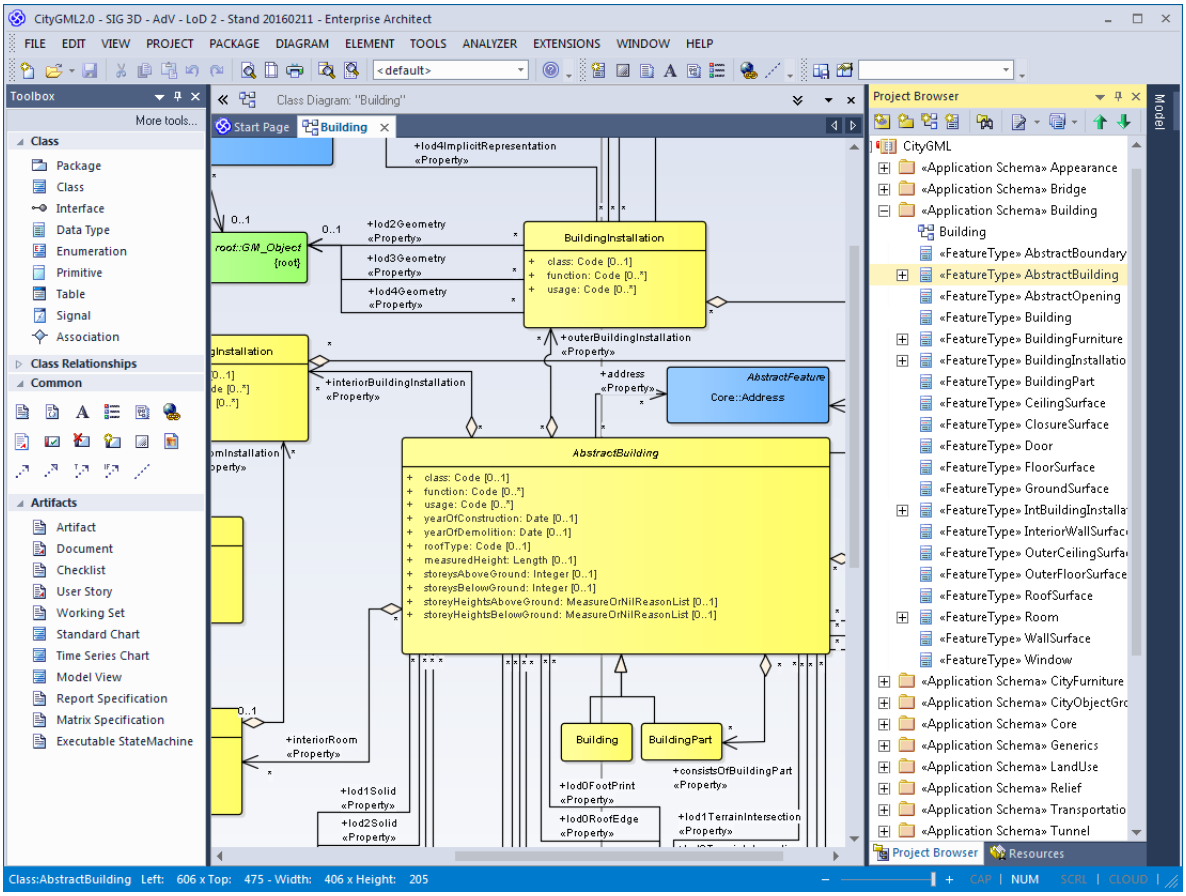

Abbildung 15: EA-Projekt Programmoberfläche

3.1.1 UML-Diagramme

In Enterprise Architect kann mit den gängigen Struktur-, Architektur-, Verhaltens- und Interaktionsdiagrammen, wie in Abbildung 16 zu sehen, gearbeitet werden. Zu den Strukturdiagrammen zählt neben dem Objektdiagramm und Paketdiagramm das Klassendiagramm (Kargl und Steinpichler 2015).

Das Klassendiagramm ist der für diese Arbeit relevante Diagrammtyp. In Klassendiagrammen werden einerseits Klassen und andererseits die Beziehungen von Klassen untereinander modelliert. Bei diesen Beziehungen wird zwischen Assoziation, Containerklasse, also Klasse in einer anderen Klasse (auch Aggregation oder Komposition genannt) und Spezialisierung bzw. Generalisierung unterschieden. Eine Klasse muss die Struktur und das Verhalten von Objekten, welche von dieser Klasse erzeugt werden, modellieren. Diese wiederum kann mit Methoden und Attributen versehen werden. Darüber hinaus ist die Modellierung von Basisklassen und Schnittstellen über Stereotypen möglich.

Die abstrakte Klasse ist die zentrale Basis für weitere Unterklassen, von der niemals Instanzen erzeugt werden. Die abstrakte Klasse wird zwar wie eine normale Klasse dargestellt, jedoch ist der Klassenname *kursiv* geschrieben. Abstrakte Klassen können mit Hilfe von Stereotypen definiert werden. In einer objektorientierten Anwendung sind Objekte die konkreten Elemente, deren Zustand durch Werte bestimmt wird, welche in den Attributen gespeichert sind (Kargl und Steinpichler 2015).

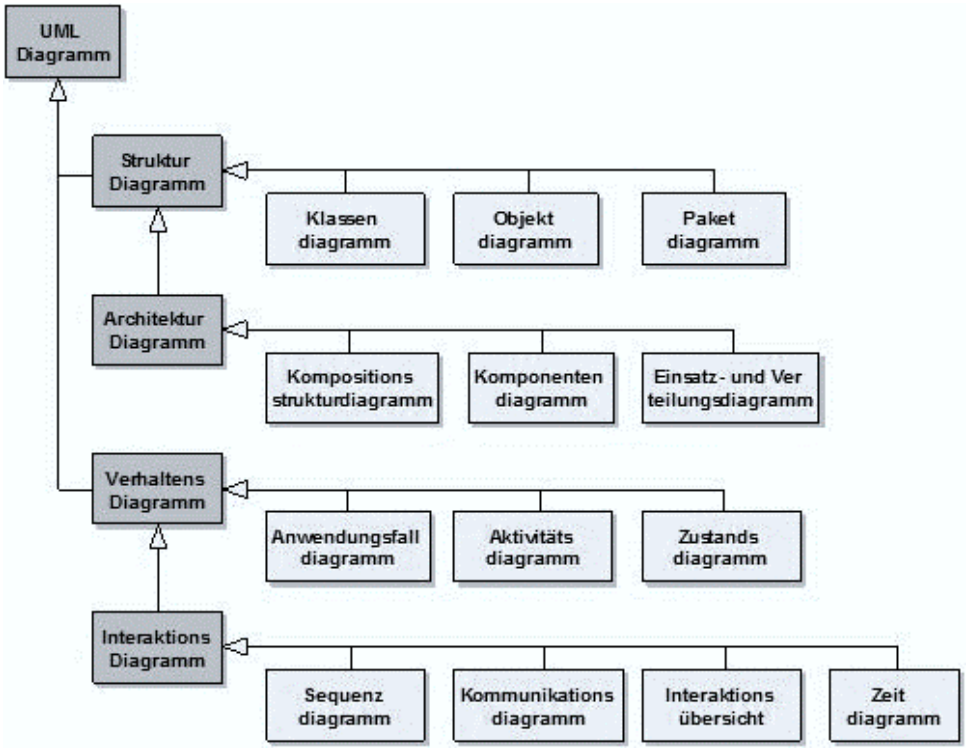

Abbildung 16: Übersicht der UML Diagramme in EA (Kargl und Steinpichler 2015)

3.1.2 Erweiterte Funktionalitäten in Enterprise Architect

Enterprise Architect 12 unterstützt weitere Funktionalitäten, die für die Bearbeitung und Weiterverarbeitung wichtig sind. Dazu zählen der Import und Export von XMI, als auch die Unterstützung von OCL. Darüber hinaus besteht die Möglichkeit erstellte Profile zu importieren als auch zu exportieren (Kargl und Steinpichler 2015).

3.1.2.1 XMI-Import und -Export

Für den Austausch der Daten wird das Datenaustauschformat XML Metadata Interchange *(*XMI*)* verwendet. XMI schließt die Lücke zwischen Objekten und XML, da XML nicht objektorientiert ist. Enterprise Architect 12 bietet umfangreiche Import- und Exportmöglichkeiten hinsichtlich des Formates XMI. In Abbildung 17 sind die aktuellen Versionen des XMI-Exports in Relation zu den gängigen UML Versionen zu sehen.

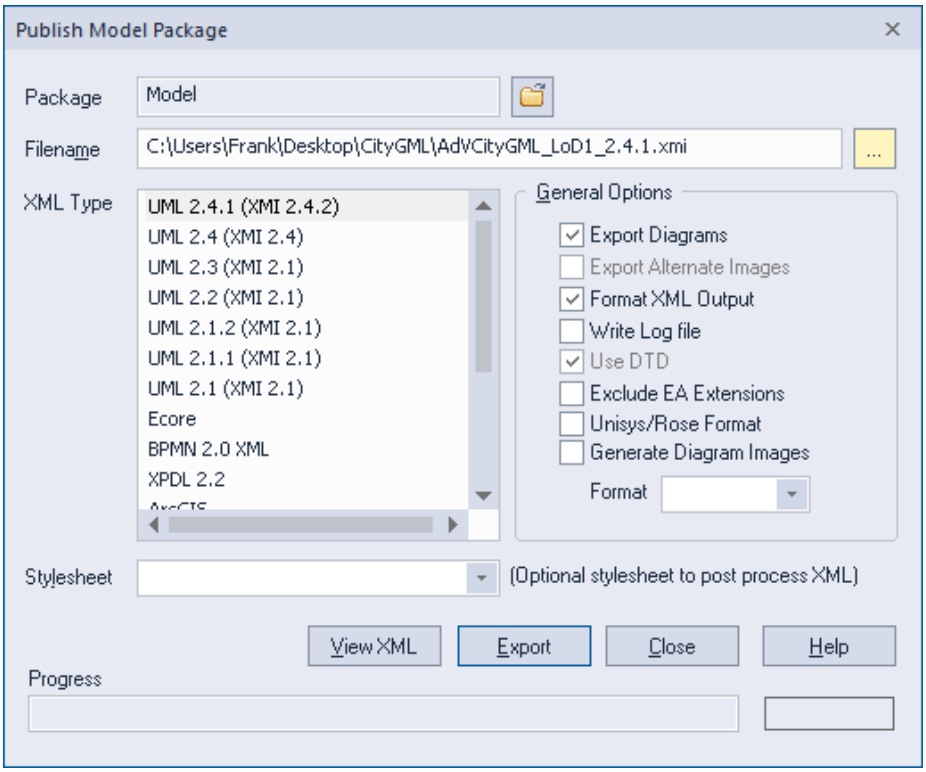

Abbildung 17: XMI-Export Dialog von Enterprise Architect

Mit der älteren Version von XMI, Version 1.0 kann es in Enterprise Architect zu Datenverlust kommen, da die 1.0 Spezifikation nicht vollständig unterstützt wird (Sparks 2012).

3.1.2.2 OCL-Unterstützung

Enterprise Architect bietet die Möglichkeit Zusicherungen, sogenannte „Constraints“ einzubringen. Dies geschieht über die bereits erwähnte OCL, mit der Ausdrücke auf UML-Modelle beschrieben werden. Man kann in Enterprise Architect OCL-Konstrukte jedem Element, jeder Beziehung oder jedem Attribut hinzufügen (Sparks 2012).
Allerdings führt Enterprise Architect lediglich eine Syntaxüberprüfung des OCL-Konstrukts durch und keine Überprüfung des Ausdrucks auf Validität. Ein OCL-Konstrukt lässt sich über das Eigenschaftenfenster eines Elements einbringen. Dabei können mehrere unterschiedliche Typen ausgewählt werden. Neben dem OCL-Konstrukt gibt es auch die Möglichkeit eine Vor- und Nachbedingung anzugeben. Damit Einschränkungen der Attribute im

Modell realisiert werden können, wird die bereits erwähnte OCL verwendet. Abbildung 18 zeigt beispielhaft die Implementierung eines OCL-Konstrukts.

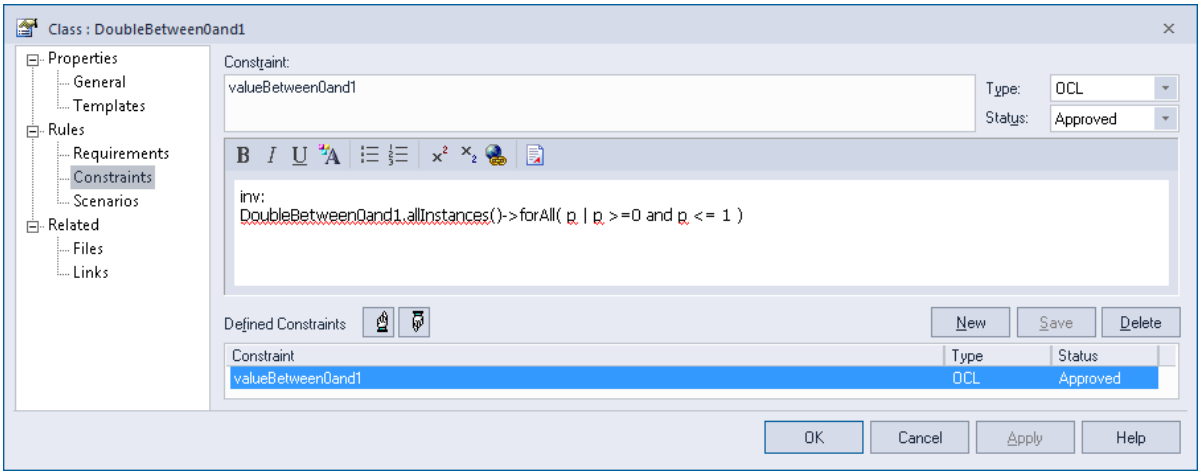

Abbildung 18: OCL-Constraint in Enterprise Architect

3.1.2.3 UML-Profile

In Enterprise Architect können UML-Profile sowohl importiert als auch exportiert werden. Der Import eines UML-Profils wird über den Project-Browser und den Tab Resources getätigt. Um in Enterprise Architect mit einem UML-Profil zu arbeiten, muss dieses über einen Rechtsklick auf „UML-Profiles“ dann auf „Import Profile“ und schlussendlich über „Filename“ die entsprechende Datei, ausgewählt und über „Import“ importiert werden. Anschließend steht das Profil, wie in Abbildung 19 veranschaulicht, zur Verfügung. UML Profile sind üblicherweise im XML-Format abgespeichert. Die Elemente stehen im Profil für das Modell zur Verfügung und es kann im Eigenschaftenfenster eines Elements dann unter Stereotypen das Profil mit den entsprechenden Stereotypen ausgewählt werden (Sparks 2012).

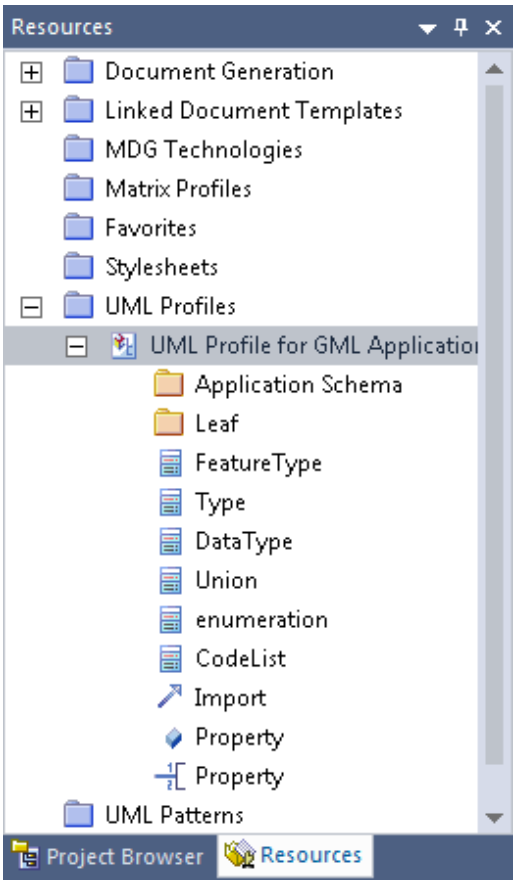

Abbildung 19: Resources-Fenster mit UML-Profil

3.2 Programmiersprache Python

Python ist eine einfach zu erlernende Programmiersprache, die sich durch eine klare Syntax und eine einfache Struktur auszeichnet, wodurch diese nicht nur lesbar, sondern auch schnell zu erlernen ist. Python bietet dennoch die Möglichkeit, komplexe Programme für unterschiedlichste Anwendungsbereiche zu entwickeln. Python-Programme werden sofort interpretiert, wodurch diese nicht erst kompiliert und gebunden werden müssen. Dies beschleunigt den Wechsel zwischen Codierungs- und Code-Testphase. Ein weiterer Vorteil ist die Unabhängigkeit vom Betriebssystem, welche ohne Neuentwicklung und Anpassung des Programmcodes auskommt (Theis 2011).

3.2.1 Modul Tkinter

Das Toolkit Tk wurde ursprünglich für die Sprache Tcl (Tool Command Language) entwickelt und steht im Modul tkinter in der Standardbibliothek von Python zur Verfügung. Damit lassen sich kleinere Anwendungen mit einer grafischen Benutzeroberfläche erstellen, für die keine zusätzlichen Bibliotheken notwendig sind. Die Entwicklungsumgebung IDLE ist ein Beispiel für ein Tk-Programm (Ernesti und Kaiser 2009).

3.2.2 Modul lxml

Um XMI-Dateien leicht zu durchsuchen, wird auf die Skriptsprache Python zurückgegriffen. Python ist geeigneter als C++/C um XML-Dokumente zu lesen. Der Begriff des Parsers ist nicht auf XML beschränkt, sondern bezeichnet ganz allgemein ein Programm, das eine Syntaxanalyse bestimmter Daten eines speziellen Formats leistet. In Python sind zwei Parser namens dom und sax enthalten, welche zwei unterschiedliche Herangehensweisen an das XML-Dokument verfolgen (Ernesti und Kaiser 2009). Aus Gründen der Performance wurde wegen der Größe der XMI-Datei auf diese beiden Varianten verzichtet.
Lxml ist eine schnelle und flexible Bibliothek für XML-Verarbeitung in Python. Lxml ist nicht wie Tkinter standardmäßig in Python enthalten, sondern muss nachträglich installiert werden. Es stellt eine ähnliche API wie ElementTree zur Verfügung und bietet zusätzliche Unterstützung für XPath, XSLT und Schema-Validierung (McNeil 2010).
Im Vergleich zu den oben genannten, bereits mit der Installation der Python Standardbibliothek enthaltenen Modulen zum Parsen von XML-Daten, ist lxml eine mächtigere und stärkere XML Bibliothek. Dieses Modul stellt eine Schnittstelle zu den in C geschriebenen Bibliotheken libxml2 und libxslt unter Verwendung der ElementTree API bereit (Lawhead 2013).
Die Entscheidung für einen geeigneten Parser für das XMI-Modell fiel aus performancegründen zugunsten des Moduls lxml aus (Behnel 2015). Lxml ist für eine effiziente Verarbeitung von sehr großen XML-Dateien und aufgrund seiner hohen Geschwindigkeit und seinem geringen Speicherverbrauch prädestiniert (Daly 2011).

Eine Analyse hinsichtlich des Speicherverbrauchs und der Geschwindigkeit unterschiedlicher Python XML-Parser findet sich unter: http://lxml.de/performance.html.

3.3 Feature Manipulation Engine (FME) als Datenprüfungswerkzeug

Die Feature Manipulation Engine ist ein Spatial-ETL-Werkzeug des kanadischen Unternehmens Safe Software. Mit dieser Anwendung sind schnelle Datenkonvertierungen und -modellierungen möglich. Verschiedenste (Geo-)Daten werden über Datenimportschnittstellen eingelesen, dabei in ein internes, neutrales Datenformat überführt und in einem gewünschten (Geo-)Datenformat ausgegeben. Auf der Basis dieses Formates können beliebige Daten zusammengeführt, verändert, geprüft oder angereichert werden. Vorhandene Datenexportschnittstellen ermöglichen es, diese Daten in ein oder mehrere Zieldatenformate zu überführen (Safe Software Inc. 2015a).
Um das Zusammenwirken von Python und FME zu gewährleisten, müssen kompatible Versionen der beiden Programme verwendet werden. Für Python liegt eine Kompatibilität mit FME lediglich bis zur Version 2.7 vor. Neuere Versionen wie die aktuelle 3.4 werden derzeit nicht unterstützt (Safe Software Inc. 2014).

3.3.1 Architektur von FME

Das zentrale Werkzeug der FME ist die FME-Workbench. Darin werden Quelldaten importiert, verarbeitet und in Zieldatenformate ausgegeben. Für die Verarbeitung von (Geo-)Daten werden sogenannte Transformer verwendet. Für jede Verarbeitungsmöglichkeit gibt es einen eigenständigen Transformer. In der grafischen Benutzeroberfläche der FME-Workbench werden anschließend die Quell- und Zieldatenformate über den Transformer verbunden, abgespeichert und/oder ausgeführt. Diesen Prozess nennt man Modellierung. Die abgespeicherte FME-Workspace lässt sich jeder Zeit wieder ausführen und musste somit nur ein einziges Mal modelliert werden. In dem abgespeicherten FME-Workspace liegen die verwendeten Daten in einem neutralen Format und die Transformationsregeln in der FME-internen Sprache vor. Mit einem einfachen Texteditor lässt sich die Datei mit der Endung **.fmw* öffnen. Dabei wird sichtbar, dass neben den Abbildungs- und Transformationsregeln in der

FME-internen Sprache spezielle Kommentare zur Erzeugung der grafischen Darstellung vorhanden sind. Der Aufbau eines FME-Workspaces besteht aus folgenden Abschnitten:

1. Pfade zu den Quell- und Zieldaten, sowie Pfad des erzeugten FME-Workspaces
2. Ausgestaltung (Lage der Reader, Writer, Transformer), sowie Lesezeichen und Kommentare
3. Bezeichnung der statischen Transformer

3.3.2 Reader und Writer

In FME werden die Quelldatensätze als Reader und die Zieldatensätze als Writer bezeichnet. Diese können datei- oder datenbankbasiert sein. Eine Liste über der unterstützten Formate für Reader und Writers findet sich unter: https://support.safe.com/knowledgedocumentation.

3.3.3 Transformer

Die unterschiedlichen Geoverarbeitungsprozesse werden durch unterschiedliche Prozesswerkzeuge, den sogenannten Transformern, durchgeführt (Safe Software Inc. 2008, 2007). Eine Liste mit Transformern findet sich unter: https://www.safe.com/transformers/#/.
Für die Arbeit mit Attributen und Attributwerten sind die Transformer Tester und TestFilter relevant. Der Tester wertet einen oder mehrere Tests auf einem Feature aus und übergibt das Feature in Abhängigkeit vom Ergebnis des Tests. Hierfür stehen zwei Ausgabe-Ports parat. Einen für die erfolgreiche und einen für die nicht erfolgreiche Testdurchführung.
Der TestFilter filtert Features anhand von Testbedingungen und richtet sie an einen oder mehrere Ausgabe-Ports (con terra GmbH 2015). Dadurch können mehrere Tests statt mit mehreren Testern mit nur einem TestFilter durchgeführt werden.

3.3.4 Factories und Functions

An dieser Stelle wird die FME-interne Sprache aufgegriffen, welche für die Umsetzung wichtig ist. In der Feature Manipulation Engine werden interne Verarbeitungsprozesse, bestehend aus Functions und Factories, sequentiell abgearbeitet (Safe Software Inc. 2008, 2015b).

FME-Workbench-Dateien können laut Tilch (Tilch 2008) als Mapping-Files angesehen werden, welche durch spezielle Kommentare zur Erzeugung grafischer Repräsentationen erweitert werden. In seiner Arbeit beschreibt er darüber hinaus den Aufbau von Mapping-files näher, während in der vorliegenden Arbeit nur der FME-Workspace ausführlicher erklärt werden soll.

FME-Workbench-Dateien bestehen durchgehend aus einem Input und einem Output an Datenströmen, welche durch einen Transformer verbunden sind (Safe Software Inc. 2015b, 2008). Diese sequentiellen Prozesse sind für Konvertierung und Qualitätssicherung der Geodaten verantwortlich. Diese FME-Workspaces dienen als Grundlage für die Stapelverarbeitung und können automatisiert oder per Prozessaufruf nutzbar gemacht werden (Safe Software Inc. 2008, 2007).

Für die Überführung von Quelldaten in ein anderes Format werden diese über einen Reader eingelesen und über einen Writer in ein gewünschtes Zieldatenformat geschrieben. Verbunden werden die Reader und Writer über Transformer, welche die gewünschten Verarbeitungsschritte durchführen (Safe Software Inc. 2009b). Der Anwender modelliert den gesamten Transformationsprozesses grafisch. Mit den Transformern werden zum Beispiel Geodaten mit externen Sachinformationen angereichert und Datensätze oder Objekte zusammengefügt. Die Modellierung erfolgt in einem grafischen User-Interface, während im Hintergrund die Transformations- und Abbildungsregeln in der internen Sprache von FME formuliert werden. Eine modellierte Transformation lässt sich mit Hilfe des Texteditors öffnen und ist menschenlesbar. Darin sind neben den beschriebenen Datenflüssen der Dateneingangs-, Datenausgangs und Verarbeitungsprozesse auch Kommentare zur grafischen Darstellung gegeben, die innerhalb des FME-Workspaces die Ausrichtung von Reader (Eingangsdatensatz), Writer (Ausgangsdatensatz) und Geoprozesswerkzeuge (Transformer) beschreiben (Safe Software Inc. 2009b).

Der Aufbau eines FME-Workspaces besteht grundlegend aus Features, Attributen und Attributtypen. Was in UML auf konzeptioneller Ebene die Klassen sind, sind in FME auf physischer Ebene die Feature types.

In FME spricht man von Attributen. Diese werden unterteilt in formatspezifische, generische FME Attribute und benutzerdefinierte Attribute (Safe Software Inc. 2007). Wie der Name

schon sagt, sind formatspezifische Attribute abhängig von einem bestimmten Format. Diese werden explizit im *FME Reader and Writer Manual* (Safe Software Inc. 2009b) beschreiben. Generische FME Attribute besitzen das Präfix „*fme*_“ und werden in fme_geometry und fme_type unterschieden. Diese Geometriedaten eines Features werden in diesen FME-internen Attributen abgelegt.
Benutzerdefinierte Feature-Attribute sind abhängig vom jeweiligen Format. Eine genaue Erklärung findet sich in *FME Reader and Writer Manual* (Safe Software Inc. 2009b).
Die in FME verfügbaren Attributtypen sind primitive Typen, wie Ganzzahlen, Gleitkommazahlen, sowie Zeichenketten. Diese werden vom Quell- und Zieldatenformat unterstützt. Als internes Datenformat werden diese Attribute als Zeichenketten abgelegt und je nach Anwendungsbereich automatisch in Zeichenketten oder numerische Werte umgewandelt (Safe Software Inc. 2007).
Die Konfiguration von *Factories* erfolgt über zusammengesetzte, aneinander gereihte Codezeilen, welche eine Kombination aus einem dem zugeordneten *INPUT, OUTPUT* und dem/den zugehörigen Transformern (Abbildung 20) mit den zugehörigen Operationen darstellt. Die Konfigurationsparameter der einzelnen Factories können aus dem Manual Features and Factories (Safe Software Inc. 2008) entnommen werden.

Syntaxregel (TestFilter):

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleBodenhoehe_TestFactory_1 INPUT FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERINPUTLINE_1 TEST @EvaluateExpression(FDIV,STRING_ENCODED,<at>Value<openparen>DatenquelleBodenhoehe<closeparen>,TestFilter_DatenquelleBodenhoehe) NOT_IN 1000<comma>1100<comma>1200<comma>1300<comma>1400<comma>1500<comma>1600<comma>1700<comma>1800<comma>2000<comma>3000<comma>4000 ENCODED BOOLEAN_OPERATOR OR COMPOSITE_TEST_EXPR <Unused> OUTPUT PASSED FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_<at>Value<openparen>DatenquelleBodenhoehe<closeparen><space>NOT_IN<space>1000<comma>1100<comma>1200<comma>1300<comma>1400<comma>1500<comma>1600<comma>1700<comma>1800<comma>2000<comma>3000<comma>40

00 OUTPUT FAILED FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERIN-PUTLINE_2

Abbildung 20: Transformationsprozess in der FME-Workbench (Safe Software Inc. 2007)

Das Input- und Output-Interface (Abbildung 21) steuert die Features, die von einer Factory übernommen und an eine andere Factory übergeben werden Die *Feature Processing Unit* ist das Verarbeitungszentrum einer Factory. Diese nimmt die übergebenen Features entgegen, verarbeitet diese je nach Vorgabe des Transformers und erzeugt daraus neue Features. Der auftretende Datenfluss wird dabei durch die Pfeile angezeigt. Wenn komplexere Prozesse mit mehreren Transformern durchgeführt werden, dann können Factories mit Attribut- und Feature-Funktionen hintereinander kombiniert und zu einer Feature Factory Pipeline (Abbildung 22), also einer Hintereinanderschaltung von Feature Factories zusammengesetzt werden.

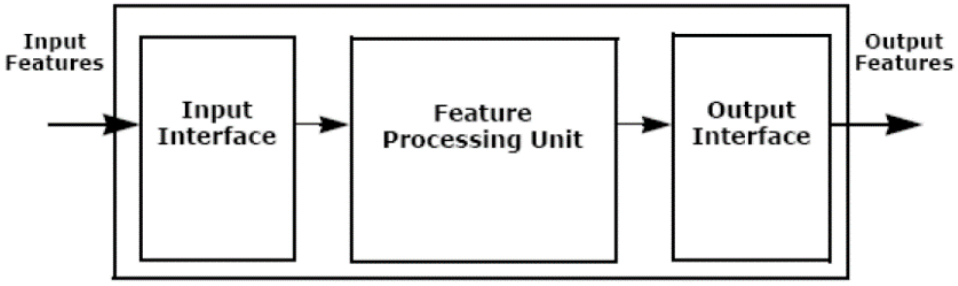

Abbildung 21: Architektur einer Factory (Safe Software Inc. 2007)

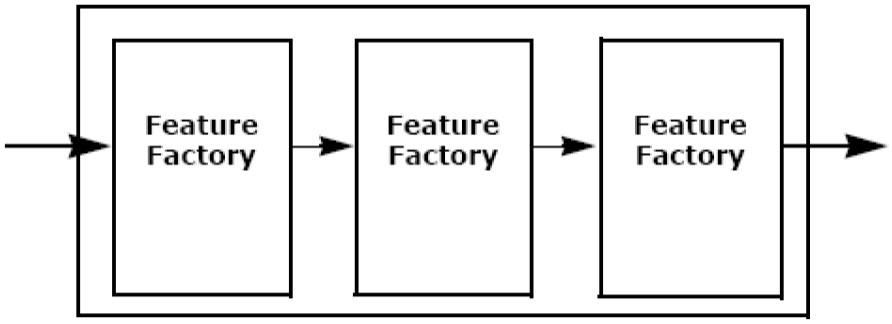

Abbildung 22: Factory Pipeline (Safe Software Inc. 2007)

Die Reihenfolge der Abarbeitung der Feature Factories läuft sequentiell, wodurch festgelegt ist, dass der Datenfluss der Features bei der ersten Feature Factory beginnt und mit der letzten Feature Factory endet. Die unterschiedlichen Factories werden demzufolge Zeile für Zeile interpretiert.

Syntaxregel (TestFilter):

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_...

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_...

FACTORY_DEF * TeeFactory FACTORY_NAME TestFilter_...

4 Konzept zur semantischen Konformitätsprüfung

Im zweiten Kapitel wurde nach einer Einführung in die theoretischen Grundlagen der Datenmodellierung mit verschiedenen Standards der OMG auf das CityGML-Modell eingegangen. Daraufhin wurde das CityGML-Modell und das CityGML-Profil der AdV in Relation gebracht. Darin wurde der Unterschied der beiden Datenmodelle genauer erläutert. Diese Unterschiede beziehen sich auf Attribute, als auch mögliche Attributwerte, welche diese Attribute annehmen können. Mit den Standards der OMG ist eine Erstellung der Anforderungen des AdV-CityGML-Profils auf Modellebene möglich.

In diesem Kapitel soll nun das Konzept zur Realisierung der Einbringung der Qualitätsangaben und der gültigen Wertebereiche verdeutlicht werden. Dabei wird genauer erklärt, wie die eingebrachten Attribute und Attributwerte in dem Modell realisiert werden und über den Export zur weiteren Verwendung vorliegen.

Im ersten Teil wird auf die Anforderungen des AdV-CityGML-Profils eingegangen. Diese Anforderungen entsprechen dem Prüfplan, der durch die AdV herausgegeben wurde. Auf Modellebene werden die Anforderungen schließlich umgesetzt und die notwendigen Attribute sowie die möglichen Attributwerte in das Datenmodell implementiert.

Anschließend wird die Modellierung der Prüfregeln innerhalb des AdV-CityGML-Profils der Modellierung weiterer Prüfregeln innerhalb der FME-Workbench gegenübergestellt.

Dabei wird aufgeführt, welche Prüfregeln umgesetzt und welche nicht umgesetzt und lediglich in FME auf Datenebene modelliert werden konnten. Die Anforderungen der AdV sollen

in diesem Kapitel konzeptionell und im darauffolgenden Kapitel prototypisch umgesetzt werden.

4.1 Anforderungen an das CityGML-UML-Datenmodell

Für den Anwendungsbereich des AdV-CityGML-Modells wurden von der AdV folgende Dokumente zur näheren Beschreibung herausgegeben.

- AdV-CityGML-Profile für 3D-Gebäudemodelle - Ergebnisse der PG „3D-Gebäudemodelle“ (AdV 2015a)
- Prüfplan für Gebäudemodelle LOD1 / LOD2 (AdV 2015d)

Anhand der Codelisten aus dem AdV-CityGML-Profil bzw. von der Internetseite der AdV (http://repository.gdi-de.org/schemas/adv/citygml/Codelisten) und den Prüfnummern für die einzelnen Attributdaten wurde von der AdV ein Prüfplan (http://www.adv-online.de/AdV-Produkte/Standards-und-Produktblaetter/Beispielsammlungen/) entwickelt. Dieser Prüfplan beinhaltet unter anderem die Pflichtattribute und die gültigen Attributwerte der Codelisten. Aus dem Dokument „AdV-CityGML-Profile für 3D-Gebäudemodelle“ geht die Zuordnung der zusätzlichen Attribute auf die Detaillierungsstufen LOD1 und LOD2 hervor.
Ziel der vorliegenden Arbeit ist die Umsetzung dieser Prüfregeln auf Modellebene und letztlich die mögliche Umsetzung auf Datenebene in Form einer Prüfsoftware. Die prototypische Umsetzung wird im nachfolgenden Kapitel genauer beschrieben.

Ziel dieses Kapitels ist unter anderem die Ergänzung der Einschränkungen der gegenüber dem CityGML-Modell zusätzlichen Attribute laut AdV-Profil im Modul *Generics*, als auch die Einschränkung der bereits existenten Attribute um OCL-Konstrukte in den Modulen *Building* und *Generics*.

Um den erwähnten Anwendungsbereich innerhalb des CityGML-Modells modellieren zu können, wird das CityGML-Modell als UML-Modell benötigt. Die Umsetzung innerhalb der

Modellebene wird in dem von der Technischen Universität München zur Verfügung gestellten CityGML-UML-Modell geschehen. Das CityGML-UML-Modell wurde von der AED-SICAD im Auftrag des Amts für Geoinformationswesen der Bundeswehr (AGeoBw) entwickelt. Dieses basiert auf der CityGML-Spezifikation 2.0 und ist gleichzeitig konform zu den erforderlichen ISO-Normen der ISO 191xx-Normenfamilie für den Geoinformationsbereich. CityGML-Anwendungsschemata können damit automatisch aus dem konzeptuellen UML-Modell unter Verarbeitung der zuvor genannten Normen abgeleitet werden. Der Lehrstuhl für Geoinformatik der Technischen Universität München (TUM) hat dieses UML-Modell im Auftrag des AGeoBW qualitätsgesichert und am 20. September 2013 zusammen mit der AGeoBW und AED-SICAD zur weiteren Verwendung der SIG3D überlassen (Kolbe et al. 2014). Das CityGML-UML-Modell ist unter: ftp://ftp.lrz.de/transfer/DidVfCGML/ zu beziehen. Das AdV-CityGML-Profil basiert auf der Reduktion auf das CityGML 1.0-Schema. Für diese Arbeit sind nicht alle Module relevant, weshalb dies vernachlässigt werden kann. Als Anforderungen werden hier die notwendigen Module genannt, damit letztendlich die für den genannten Anwendungsbereich notwendigen zusätzlichen Einschränkungen für die Attribute eingebracht werden sollen. Diese Module des CityGML-Datenmodells sind *Generics* und *Building*.

4.2 Ergänzung des CityGML-UML-Datenmodells um OCL-Regeln

In das Modul *Generics* müssen die OCL-Regeln für die Qualitätsangaben eingebracht werden. Für diese Arbeit werden, ausgehend von dem Prüfplan der AdV, folgende Einschränkungen in das CityGML-UML-Modell eingebracht: *Gemeindeschluessel, DatenquelleDachhoehe, DatenquelleBodenhoehe, DatenquelleLage* und *BezugspunktDach*, welches nur im LOD1 vorkommt. Die Attributdaten der generischen Attribute entsprechen der Prüfnummer 2220 im Prüfplan. Für die Detaillierungsstufen LOD1 und LOD2 werden die generischen Attribute in Enterprise Architect als OCL-Constraints in das Modul *Generics* eingebracht. Abbildung 23 veranschaulicht die OCL-Konstrukte für die Detaillierungsstufe LOD2 am Beispiel des Attributes *Gemeindeschluessel*. Dem gegenüber veranschaulicht Abbildung 24

die Diagrammansicht dieser eingebrachten OCL-Konstrukte, welche grafisch in Enterprise Architect in das UML-Modell eingebracht wurden.

Die Bezeichnungen der eingebrachten Attribute in OCL müssen der Schreibweise von CityGML in der Interpretation von FME entsprechen, damit diese für die Erstellung des FME-Workspaces die Prüfung der Attribute korrekt durchführt. Als Beispiel wäre hier das Attribut *function* zu nennen, welches in der FME-Workbench als *citygml_function* bezeichnet wird.

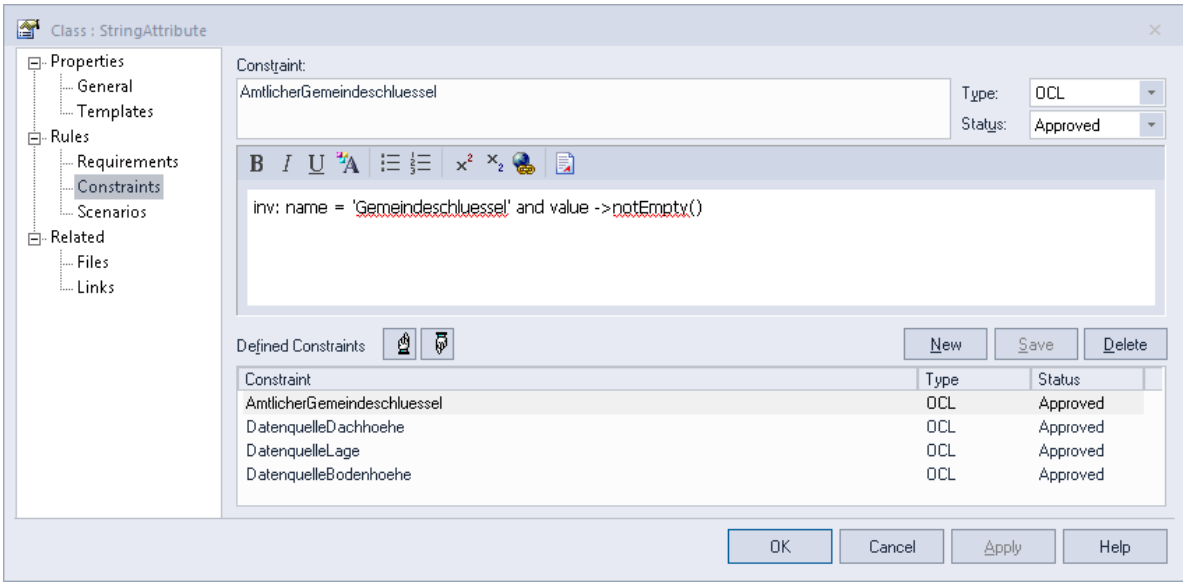

Abbildung 23: Einbringung der OCL-Konstrukte in die Klasse StringAttribute im Modul Generics für das LOD2

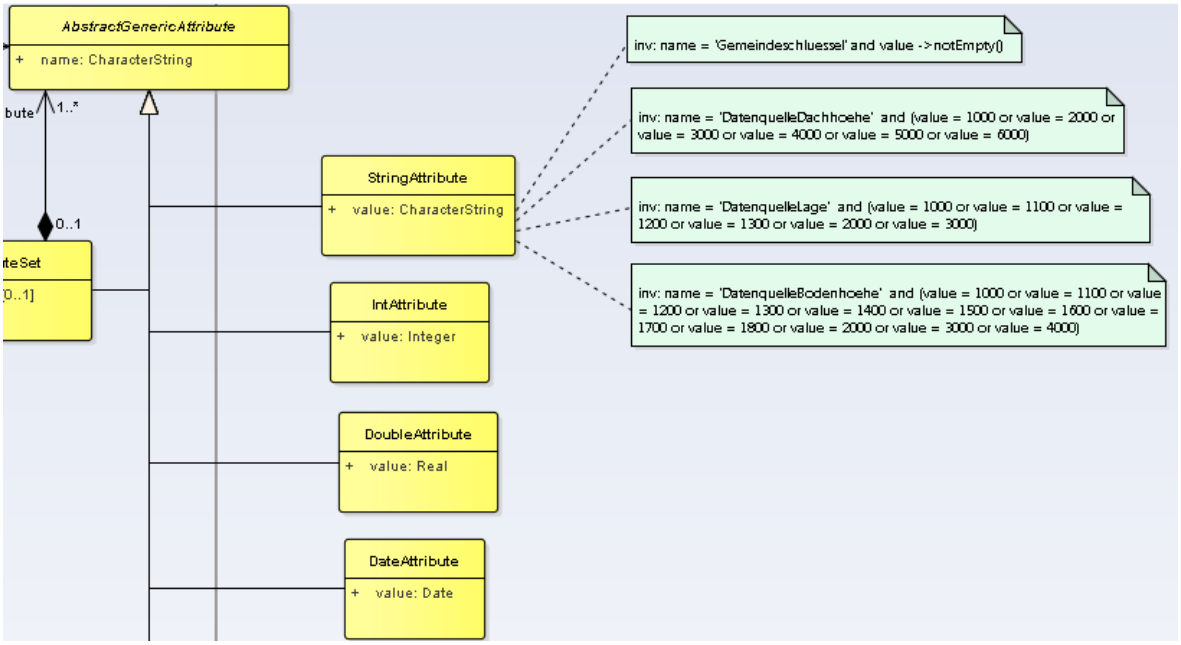

Abbildung 24: Grafische Darstellung der OCL-Konstrukte in Enterprise Architect für das LOD2

Den generischen Attributen stehen die Attribute gegenüber, die sich im Modul Building befinden. Für diese müssen weitere OCL-Regeln eingebracht werden. Diese Attribute lauten für das LOD2 *Function* (Prüfnummer 2230) für das Attribut *function*, *Lagebezeichnung* (Prüfnummer 2270) für das Attribut *addressFeature*, *ExternalReference* (Prüfnummer (5000) für das Attribut *externalReference*, *Gebauedename* (Prüfnummer 2430) für das Attribut *gml:name*, *Ableitungsdatum* (2210) für das Attribut c*reationDate* und *Objektidentifikator* (2210) für das Attribut *gml:id.*. Da ALKIS noch nicht flächendeckend in Deutschland eingeführt ist, wird die Referenz derzeit nicht in allen Datensätzen geführt. Für die Umsetzung des Prüfplans wurde diese dennoch in das Datenmodell integriert. Die OCL-Konstrukte der genannten Attribute wurden in die Klasse Building eingebracht, wie in Abbildung 25 zu sehen.

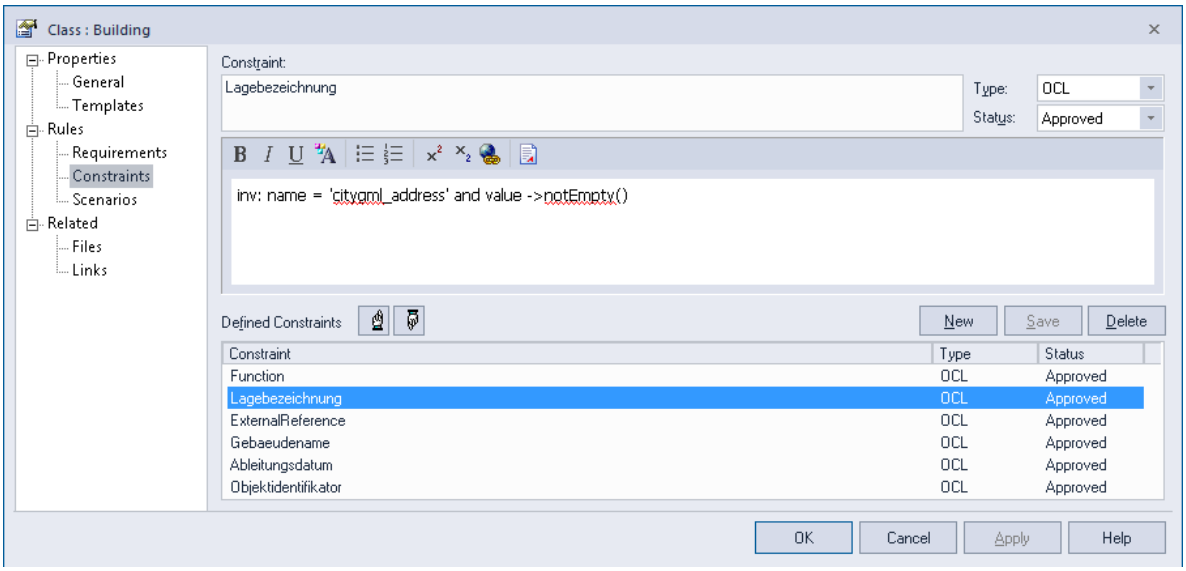

Abbildung 25: Einbringung der Attribute in die Klasse Building (LOD2)

Analog zur Einbringung der OCL-Regeln in die Klasse Building werden OCL-Regeln für die Attribute *roofType* (Prüfnummer 2240), *measuredHeight* (Prüfnummer 2250) und *storeysAboveGround* (Prüfnummer 2270) in die Klasse BuildingPart (Abbildung 26) eingebracht.

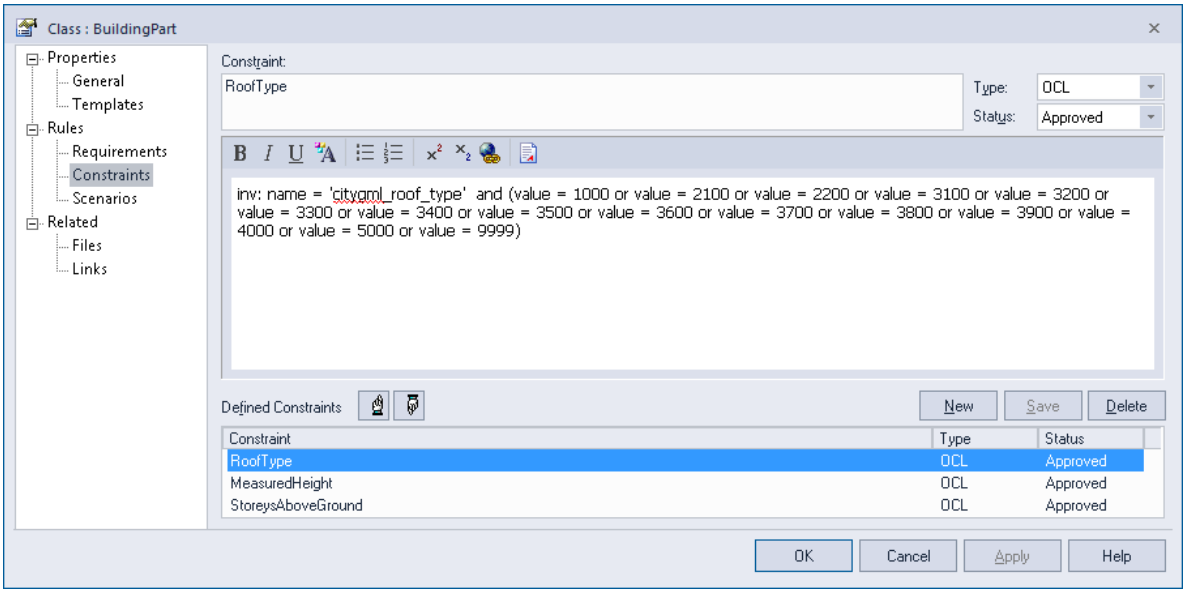

Abbildung 26: Einbringung der OCL-Konstrukte in die Klasse BuildingPart des Moduls Building

Für das Attribut *Function*, welches sich im Modul *Building* in der Klasse *Building* befindet, wurden die gültigen Attributwerte (Prüfnummer 2230) laut AdV umgesetzt. Die gültigen Wertebereiche sind unter: http://repository.gdi-de.org/schemas/adv/citygml/Codelisten/BuildingFunctionTypeAdV.xml abgelegt. In Abbildung 27 ist die Umsetzung dieser Attributwerte abgebildet.

Das Attribut *ExternalReference* wurde in das mit Hilfe von OCL und der Einschränkung nicht leer sein zu dürfen in das Datenmodell eingebracht. Es lässt sich jedoch nicht prüfen, da es sich bei diesem Attribut um mehrere Attribute handelt, die zusammengesetzt sind. Die einzelnen Features werden in FME als *externalReference{}.externalObject.name*, *externalReference{}.external.Object.uri* und *externalReference{}.informationSystem* aufgelistet. Die geschweifte Klammer „*{}*“ zeigt an, dass es sich um eine Liste handelt.

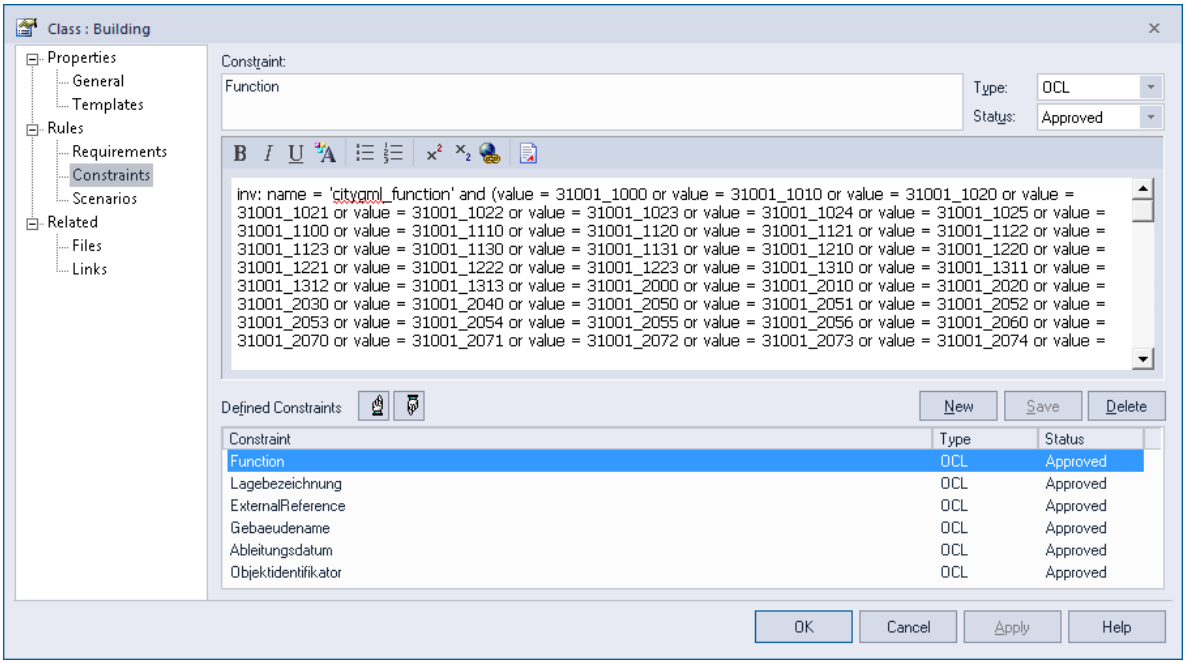

Abbildung 27: Einbringung der OCL-Konstrukte der Attribute in die Klasse Building im Modul Building

Neben den Attributen, welche durch eine Codeliste definiert werden, sind weitere Attribute im AdV-CityGML-Profil enthalten, welche lediglich auf Existenz eines Attributwertes geprüft werden können.

Die Umsetzung der Prüfregeln des Prüfplans der AdV ist die zentrale Aufgabe dieser Arbeit. Die Prüfnummern sind aus dem Prüfplan zu entnehmen.

In Anhang A sind die OCL-Konstrukte der Attribute hinterlegt. Diese unterscheiden sich hinsichtlich der Überprüfung auf Existenz, einem gültigen Wert aus der Codeliste der AdV und auf einen Attributwert größer Null.

Tabelle 5 zeigt die Prüfnummern aus dem Prüfplan der AdV, welche Prüfregeln in OCL definierbar sind und welche nicht. Daraus lässt sich ebenfalls ableiten, dass bestimmte Attribute zwar nicht in OCL, aber eventuell in FME definierbar sind. Darauf wird im übernächsten Abschnitt eingegangen.

Prüfnummer	**Testkriterien**	**definierbar in OCL**
2100	Keine leeren Attribute	ja
2210	Attribute vollständig?	nein
2220	Korrekte Werte aus der Codeliste generische Attribute	ja
2221	Korrekte Belegung des Attribut Gemeindeschlüssel	nein
2230	Korrekte Werte aus der Codeliste - function	ja
2240	Korrekte Werte aus der Codeliste - rooftype	ja
2250	minimale Höhe	ja
2260	maximale Höhe	ja
2270	Attribute dem Gebäude oder Gebäudeteil zugeordnet	nein
2280	länderspezifische Attribute vorhanden	nein
2310	BuildingPart gehört nur zu einem Gebäude	nein
2320	BuildingParts Teil eines Gebäudes	nein
2330	keine Unterteilung von BuildingParts	nein
2410	Korrekte OID?	nein
2420	Ist der Dateiname korrekt?	nein
2430	<gml:name> bei Citymodel vorhanden? Sind die Werte zulässig	nein
2510	Pro CityGML-Datei ist ein Envelope zu bilden	nein
2520	CRS nur in Envelope bei CityModel?	nein

2610	Alle Geometrien mit 3 Nachkommastellen? Auch das Envelope	nein
2620	keine Kreisbögen im Datensatz	nein
2630	Keine Referenzen auf Geometrie anderer Objekte	nein

Tabelle 5: In OCL abbildbare Prüfregeln der AdV

Wie aus Tabelle 5 ersichtlich, ist nur ein geringer Teil der Prüfregeln des Prüfplans der AdV in OCL umsetzbar. Generell ist nur eine Überprüfung auf Existenz eines Attributwertes *notEmpty()*, sowie auf die Werte aus der Codeliste möglich. In Tabelle 6 wird auf diejenigen Prüfnummern mit einer Begründung näher eingegangen, welche nicht in OCL ausdrückbar sind und warum eine Umsetzung nicht möglich ist. Für die Attribute *measuredHeight* und *storeysAboveGround* wurde eine Einschränkung vorgenommen, dass diese größer Null sein müssen. Dies stellt zwar keine direkte Umsetzung des Prüfplans dar, jedoch eine Möglichkeit, die Attributwerte einer logischen Einschränkung zu unterziehen.

Prüfnummer	**Nicht umsetzbar, weil…**
2210	…eine Prüfung, ob ein Attribut existiert, mit OCL nicht möglich ist. Die Prüfung wurde abgeändert hinsichtlich eines Attributwertes
2221	… mit OCL keine inhaltliche Prüfung möglich ist, ob der Attributwert bestimmte Zeichen enthält.
2270	… mit OCL es nicht möglich ist zu überprüfen, ob ein Attribut dem Gebäude oder Gebäudeteil zugeordnet ist.
2280	… mit OCL es nicht möglich ist zu überprüfen, ob andere Attribute, als die AdV-spezifischen Attribute vorhanden sind.
2310	… mit OCL es nicht möglich ist zu überprüfen, ob ein BuildingPart nur zu einem Gebäude gehört.

2320	… mit OCL es nicht möglich ist zu überprüfen, ob ein Gebäude 0 oder mehr als ein BuildingPart besitzt.
2330	… mit OCL es nicht möglich ist zu überprüfen, dass ein Gebäudeteil nicht in weitere Gebäudeteile unterteilt ist.
2410	… mit OCL es nicht möglich ist zu überprüfen, ob der Objektidentifikator das korrekte Bundeslandkürzel enthält.
2420	… mit OCL es nicht möglich ist zu überprüfen, ob der Dateiname den korrekten Detaillierungsgrad und das korrekte Bundeslandkürzel enthält.
2430	… mit OCL es möglich ist zu überprüfen, ob der gml:name bei Citymodel vorhanden ist, jedoch nicht, ob die Werte zulässig sind. Hierzu zählen der korrekten Detaillierungsgrad und das korrekte Bundeslandkürzel.
2510	…OCL nicht die Prüfung zulässt, ob pro CityGML-Datei ein Envelope gebildet wurde.
2520	…OCL nicht die Prüfung zulässt, dass sich die Angabe des Koordinatensystems nur im Envelope befindet.
2610	…mit OCL keine Überprüfung vorgenommen werden kann, dass alle Geometrien maximal 3 Nachkommastellen haben.
2620	…mit OCL keine Überprüfung vorgenommen werden kann, dass keine Kreisbögen im Datensatz vorhanden sind.
2630	…weil mit Hilfe von OCL sich nicht einschränken lässt, dass keine Referenzen auf Geometrien anderer Objekte vorhanden sind.

Tabelle 6: Nicht in OCL umsetzbare Prüfnummern des AdV-Prüfplans

Neben der Überprüfung auf Attributwerte aus einer Codeliste ist eine Überprüfung auf Existenz eines Attributwertes möglich. In dieser Arbeit wurde lediglich eine der beiden Überprüfungen in OCL umgesetzt, wenn eine Codeliste besteht. Darüber hinaus wurde für die Attribute *measuredHeight* und s*toreysAboveGround*, welche nicht durch den Prüfplan der AdV definiert sind, wie bereits erwähnt OCL-Konstrukte in das UML-Modell eingebracht. Die Syntax richtet sich bei diesen Attributen nach dem Namen, welche in FME zugewiesen wird. Für das Attribut *measuredHeight* lautet dieser Name *citygml_measured_height.*
Das OCL-Konstrukt lautet folglich:

inv: name = 'citygml_measured_height' and value > 0

Grundsätzlich ist aber auch eine Umsetzung beider Überprüfungen möglich. Die OCL-Syntax sieht wie folgt aus:

inv: name = 'citygml_roof_type' and value ->notEmpty() and (value = 1000 or value = 2100 or value = 2200 or value = 3100 or value = 3200 or value = 3300 or value = 3400 or value = 3500 or value = 3600 or value = 3700 or value = 3800 or value = 3900 or value = 4000 or value = 5000 or value = 9999)

Neben den Attributwerten müssen auch die Operatoren auf Überführbarkeit überprüft werden. In dieser Arbeit wird dieses Thema kurz erwähnt, da Operatoren nicht überführt wurden.

4.3 Validierung und Export der OCL-Konstrukte

Wie bereits erwähnt, ist es nicht möglich, OCL-Constraints in Enterprise Architect zu validieren. Die Möglichkeit der Validierung von OCL-Constraints in Enterprise Architect bezieht sich nur auf die OCL-Syntax, jedoch nicht auf die inhaltliche Korrektheit des Ausdrucks bezüglich der Attributwerte.

Das CityGML-Modell, welches nach Vorgaben der AdV in Enterprise Architect erstellt wurde, wird als XMI exportiert. Hier stehen einige unterschiedliche Versionen zur Verfügung. Es wird die Version 2.4.1 verwendet, welche auch die aktuellste laut ISO-Spezifikation darstellt. In dieser Version sind die eingebrachten OCL-Konstrukte unter „<packagedElement>“, „<ownedRule>“ und „<specification>“ mit den Attributnamen und Attributwerten abgelegt, wie in Abbildung 28 zu sehen ist.

```
<packagedElement xmi:type="uml:Class"
      xmi:id="EAID_94F3B0A6_EE68_4efa_8E1B_8DDFA8F95FB8" name="Strin-
      gAttribute">
      <ownedRule xmi:type="uml:Constraint"
            xmi:id="EAID_OR000000_EE68_4efa_8E1B_8DDFA8F95FB8" name="Amt
            licherGemeindeschluessel">
            <constrainedElement
                  xmi:idref="EAID_94F3B0A6_EE68_4efa_8E1B_8DDFA8F95FB8"/>
            <specification xmi:type="uml:OpaqueExpression"
                  xmi:id="EAID_COE000000_EE68_4efa_8E1B_8DDFA8F95FB8"
                  body="inv: name = 'Gemeindeschluessel' and value -
                  &amp;gt;notEmpty()"/>
      </ownedRule>
      <ownedRule xmi:type="uml:Constraint"
            xmi:id="EAID_OR000001_EE68_4efa_8E1B_8DDFA8F95FB8"
            name="DatenquelleBodenhoehe">
            <constrainedElement
                  xmi:idref="EAID_94F3B0A6_EE68_4efa_8E1B_8DDFA8F95FB8"/>
            <specification xmi:type="uml:OpaqueExpression"
                  xmi:id="EAID_COE000001_EE68_4efa_8E1B_8DDFA8F95FB8"
                  body="inv: name = 'DatenquelleBodenhoehe'  and (value =
                  1000 or value = 1100 or value = 1200 or value = 1300 or
                  value = 1400 or value = 1500 or value = 1600 or value =
                  1700 or value = 1800 or value = 2000 or value = 3000 or
                  value = 4000)"/>
      </ownedRule>
      <ownedRule xmi:type="uml:Constraint"
                  xmi:id="EAID_OR000002_EE68_4efa_8E1B_8DDFA8F95FB8"
                  name="DatenquelleDachhoehe">
            <constrainedElement
                  xmi:idref="EAID_94F3B0A6_EE68_4efa_8E1B_8DDFA8F95FB8"/>
            <specification xmi:type="uml:OpaqueExpression"
                  xmi:id="EAID_COE000002_EE68_4efa_8E1B_8DDFA8F95FB8"
                  body="inv: name = 'DatenquelleDachhoehe'  and (value =
                  1000 or value = 2000 or value = 3000 or value = 4000 or
                  value = 5000 or value = 6000)"/>
      </ownedRule>
      <ownedRule xmi:type="uml:Constraint"
                  xmi:id="EAID_OR000003_EE68_4efa_8E1B_8DDFA8F95FB8"
                  name="DatenquelleLage">
            <constrainedElement
                  xmi:idref="EAID_94F3B0A6_EE68_4efa_8E1B_8DDFA8F95FB8"/>
```

```
            <specification xmi:type="uml:OpaqueExpression"
                  xmi:id="EAID_COE000003_EE68_4efa_8E1B_8DDFA8F95FB8"
                  body="inv: name = 'DatenquelleLage'  and (value = 1000
                  or value = 1100 or value = 1200 or value = 1300 or
                  value = 2000 or value = 3000)"/>
      </ownedRule>
      <ownedAttribute xmi:type="uml:Property"
                  xmi:id="EAID_4EB4414D_527A_4d2b_B684_46E93FE071FB"
                  name="value">
            <type
                  xmi:idref="EAID_AF7C81A6_B1C1_4469_A09F_B97989024A14"/>
            <lowerValue xmi:type="uml:Lit
                  eralInteger"
                  xmi:id="EAID_LI000653_527A_4d2b_B684_46E93FE071FB"
                  value="1"/>
            <upperValue xmi:type="uml:LiteralUnlimitedNatural"
                  xmi:id="EAID_LI000654_527A_4d2b_B684_46E93FE071FB"
                  value="1"/>
      </ownedAttribute>
      <generalization xmi:type="uml:Generalization"
                  xmi:id="EAID_1FB1659C_43CE_444d_9114_2C5742F17F07" gen
                  eral="EAID_BCAF630B_3B88_4629_8BB1_B2AAB721E498"/>
</packagedElement>
```

Abbildung 28: XMI-Export LOD2 („AdVCityGML_LoD2.xmi“)

Die Anzahl der (Unter-)Packages, welche das Stammpackage „ISO TC211“ beinhaltet, beträgt 45. Für die Weiterverarbeitung sind nicht alle Packages notwendig. Der Export des CityGML-AdV-UML-Modells aus Enterprise Architect wird beschleunigt, indem nur folgende Packages, wie in Abbildung 29 abgebildet, aus dem Stammpackage „ISO TC211“ weiterhin bestehen bleiben. Dazu gehören „Drafts“, „Package Context Diagrams“, „ISO 19103 Conceptual Schema Language“, „ISO 19107 Spatial Schema“, “ISO 19136 GML”, “Informative” und “Orphans”. Alle anderen Pakete sind für den Export eines validen UML-Modells für den in dieser Arbeit genannten Zweck nicht notwendig.

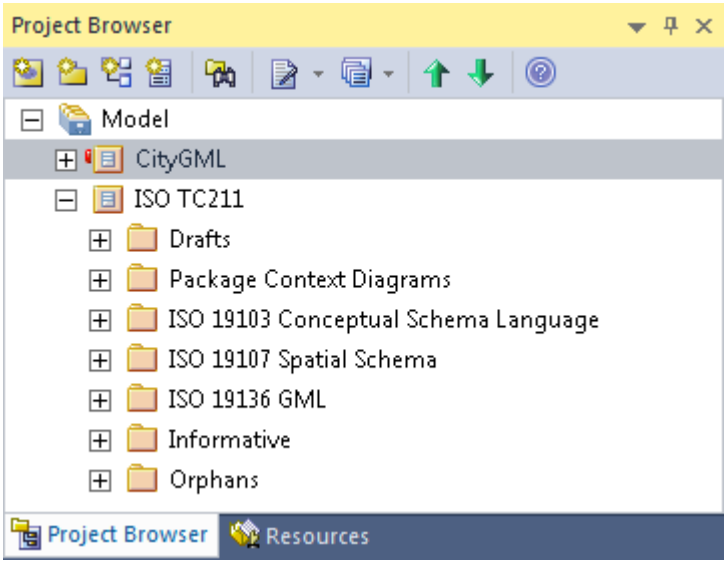

Abbildung 29: Benötigte Packages für validen Export in XMI

4.4 Umsetzung der Prüfregeln in FME

In diesem Abschnitt wird die Umsetzung der OCL-Konstrukte in dem Datenmodell näher erläutert. Es wird darauf eingegangen, welche unterschiedlichen Prüfungen vorgenommen werden und wie diese letztlich realisiert wurden. Dabei geht es im ersten Abschnitt um die benutzerdefinierten Werte für die Attribute, welche durch Codelisten der AdV vorliegen.
Der nächste Teil stellt die Überprüfung der Attribute auf Existenz dar. Hierbei wird in FME zwischen „Attribute is Empty String“ „Attribute is Null“ und „Attribute is Missing” unterschieden. Anschließend wird die Realisierung der Attributwerte größer Null aufgezeigt. Der letzte Teil stellt die Attribute dar, welche lediglich größer Null sein müssen.
Die Umsetzung der Einschränkungen der gültigen Wertebereiche innerhalb des CityGML-UML-Modells wurde im vorherigen Abschnitt dieser Arbeit thematisiert und tiefer erläutert. Im ersten Abschnitt geht es auch um die Umsetzbarkeit der Einschränkung von der Modellebene auf die Datenebene in Form der Validierung von CityGML-Datensätzen innerhalb der FME-Workbench. Die Einschränkungen, die in Enterprise Architect in der UML mit Hilfe der OCL vorgenommen wurden, konnten in dieser Arbeit bis zu einem gewissen Grad umgesetzt werden.

4.4.1 Attributwertprüfung mit Codelisten

Die Attribute, welche durch Attributwerte in Codelisten spezifiziert werden, stellen für diese Arbeit den wichtigsten Teil dar. Für die Umsetzung dieser Attributwerte ist ein geeigneter Operator innerhalb der FME notwendig, um die Einschränkungen entsprechend den Vorgaben des Prüfplans korrekt abbilden zu können.
Für die verwendeten Transformer (Tester und Testfilter) wurde für die Überprüfung der gültigen Attributwerte mit Codelisten ein entsprechender Operator gefunden. In dem negierten Operator „In“(„NOT_IN“) wurde für die Attributwerte mit Codelisten der richtige Operator ausgewählt, um nicht vorkommende Attributwerte mit dieser Liste im Prüfungsprozess abzugleichen und über einen Outputport des jeweiligen Transformers als fehlerhaft weitergeben zu können.
Die Attribute mit Codelisten in FME heißen *citygml_function*, *citygml_rooftype*, *DatenquelleBodenhoehe*, *DatenquelleDachhoehe*, *DatenquelleLage*, *BezugspunktDach* (nur im LOD1 vorkommend). Auf Modellebene wurden zuvor die Codelisten der AdV vollständig umgesetzt und liegen für die Prüfung vor. Abbildung 30 zeigt die Umsetzung der Attributwerte für das Attribut *DatenquelleBodenhoehe*. Die Umsetzung der Attribute mit Attributwerten in Codelisten war vorrangiges Ziel dieser Arbeit. Jedoch konnten zur Umsetzung des Prüfplans weitere Attributprüfungen in FME vorgenommen werden. Diese werden in den folgenden Abschnitten vorgestellt.

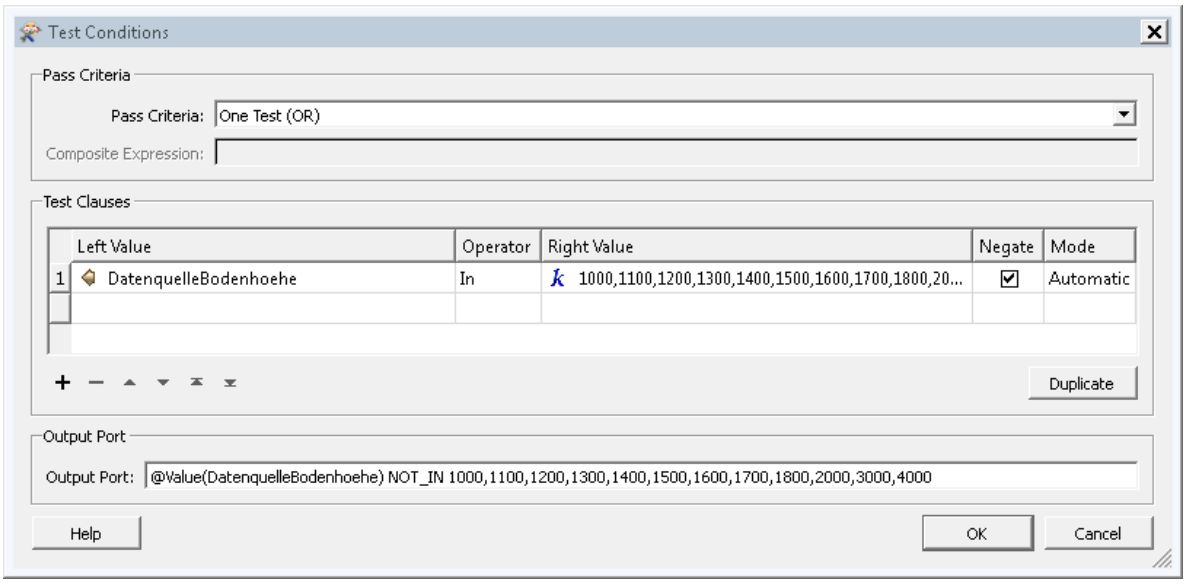

Abbildung 30: Umsetzung der Attributwerte mit Codelisten

4.4.2 Attributwertprüfung auf Existenz

Die Prüfung auf Existenz wurde aus OCL nicht überführt. Eine Umsetzung der OCL-Konstrukte nach FME ist nicht ohne weiteres möglich. In OCL wird die Prüfung über die Syntax *notEmpty()* vorgenommen, wie in Abbildung 25 zu sehen. Bei den Operatoren, welche Attributwerte auf Existenz prüfen wird in Safe Software FME zwischen „Attribute is Null“, „Attribute is Empty String” und “Attribute is Missing” unterschieden. Alle 3 haben unterschiedliche Bedeutungen. Während ein Nullattributwert als Vertreter für einen unbekannten Wert zu sehen ist, den wir nicht kennen, ist „Attribute is Empty String“ ein bekannter Wert, also quasi ein nicht existenter Wert. Ein fehlender Attributwert “Attribute is Missing” ist ein Wert, der existent sein sollte, aber fehlt. Dieser Operator wird normalerweise verwendet, um zu überprüfen, ob alle Attribute einen Wert haben, ganz gleich welchen (Safe Software Online 2015). Um alle Vorgaben des Prüfplans abbilden zu können, werden nur zwei dieser Operatoren verwendet. In Tabelle 7 sind die Prüfnummern den Operatoren aus OCL und FME gegenübergestellt. Wenn der Inhalt ausschließlich Leerzeichen, Tabulatoren und Zeilenumbrüche enthält, gilt das Attribut auch dann als leer und wird in FME mit dem Operator

„Attribute Is Empty String“ abgebildet. Der Operator „Attribute is Missing“ bezieht sich auf tatsächliche fehlende Attributwerte.

Prüfnummer im Prüfplan	Operator in OCL	Operator in FME
2100	notEmpty()	Attribute is Empty String
2100	notEmpty()	Attribute is Missing

Tabelle 7: Prüfnummern aus Prüfplan, Operatoren aus OCL und FME

Beispielhaft ist dies für den Gemeindeschlüssel in Abbildung 31 dargestellt. Wie bereits erwähnt, gilt ein Attribut auch dann als leer, wenn der Inhalt ausschließlich Leerzeichen, Tabulatoren oder Zeilenumbrüche sind (AdV 2015d).

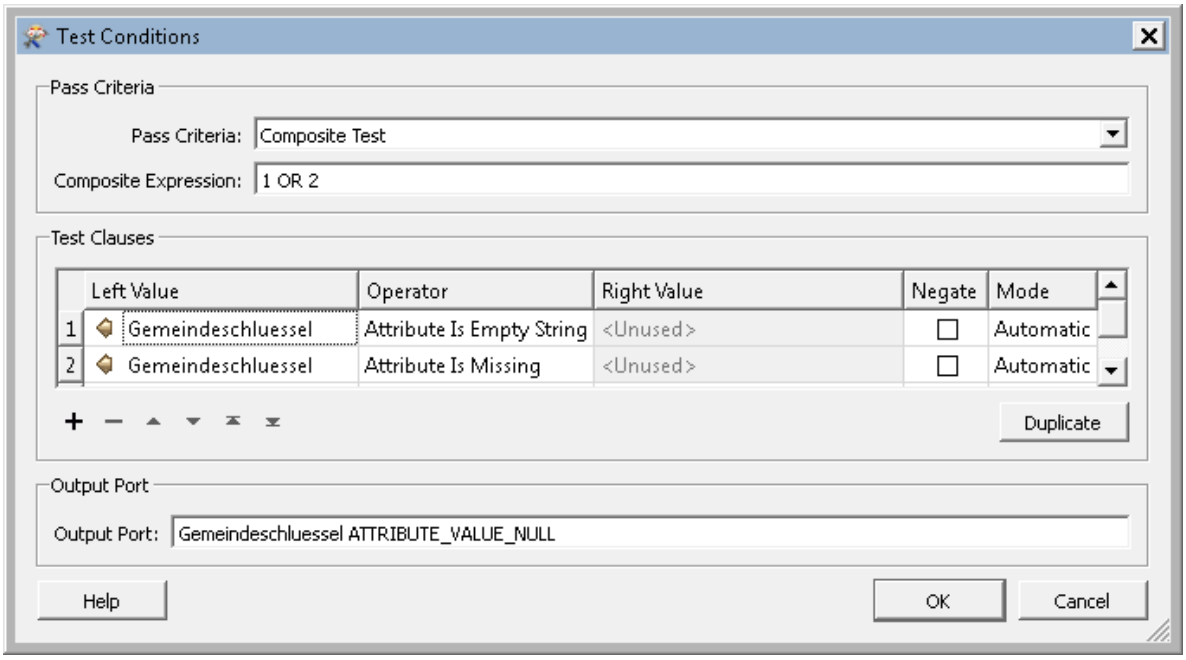

Abbildung 31: Prüfung des Gemeindeschlüssels auf Existenz des Attributwertes in FME

4.4.3 Attributwertprüfung größer Null

Die Überprüfung, ob der Attributwert größer Null ist, wurde zwar in OCL umgesetzt, jedoch nicht im Rahmen dieser Arbeit aus OCL nach FME überführt. Die Überprüfung der Attribute „citygml_measured_height“ und „citygml_storeys_above_ground” wurde nur in FME (Abbildung 32) mit dem negierten Operator „>“ und dem Wert „0“ modelliert. Zwar konnten damit nicht die genauen Vorgaben der AdV bezüglich der Genauigkeit mit 3 Nachkommastellen umgesetzt werden, jedoch war eine Prüfung, dass diese Werte positiv sein müssen, möglich. In Enterprise Architect sind diese Werte über das „>“ Zeichen modelliert. Beispielhaft sieht das OCL-Konstrukt wie folgt aus:

inv: name = 'measuredHeight' and value > 0

In der exportierten XMI-Datei sind die Werte mit „>“ angegeben. Dies weicht von den HTML Entitäten ab, die das größer-Zeichen mit „>“ angibt. Bei einer vollständigen Umsetzung der OCL-Konstrukte müsste hierbei eine Übersetzung vorgenommen werden.

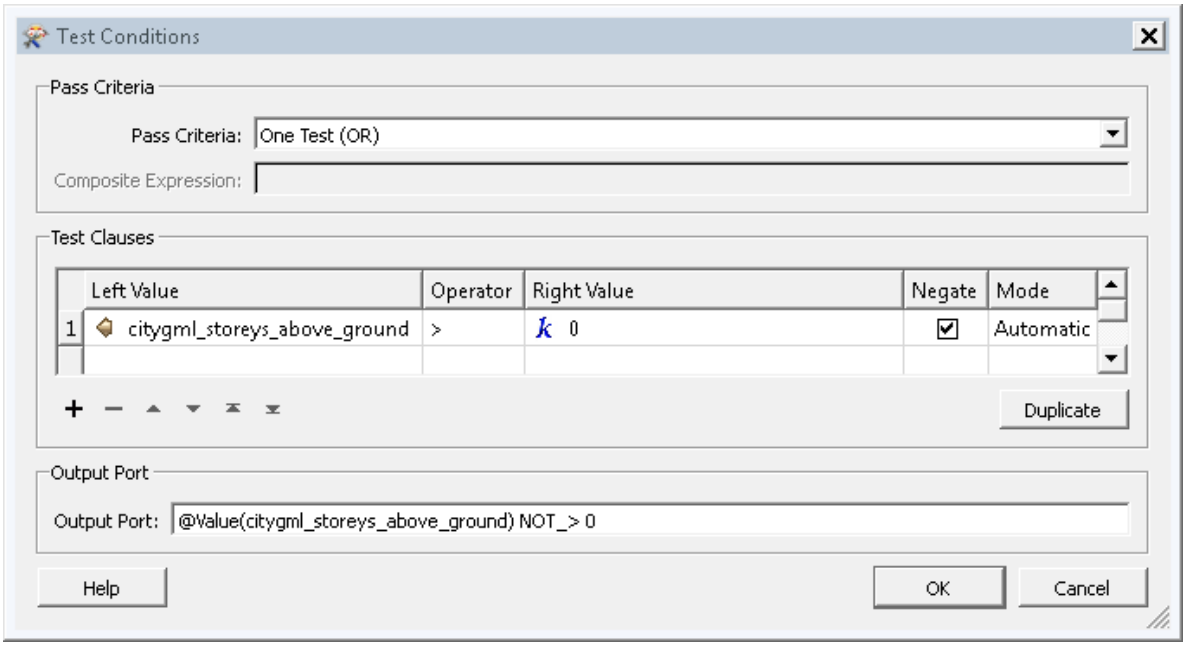

Abbildung 32: Attributwert größer Null

4.5 Umsetzbarkeit des Prüfplans in FME

Der erste Teil dieses Kapitels beschäftigte sich mit der Frage, welche Prüfregeln des Prüfplanes der AdV in OCL ausdrückbar sind, welche nicht und warum nicht. Anschließend wurde auf die Umsetzung der Prüfregeln aus den OCL-Konstrukten nach FME eingegangen und welche Prüfregeln zwar überführbar wären, aber in FME modelliert werden mussten. Dieser Teil beschäftigt sich mit der Umsetzbarkeit des Prüfplans direkt in FME, da diese Prüfregeln mit OCL nicht ausdrückbar sind.

Laut Prüfplan der AdV muss der amtliche Gemeindeschlüssel aus genau 8 Ziffern bestehen. Dies ist jedoch weder in OCL noch in FME realisierbar. Eine Prüfung, dass die erste Ziffer Null oder Eins sein muss, wurde in FME modelliert. Dies ist in der Abbildung 33 veranschaulicht. In FME wurde dies mit dem Operator „Begins With“ modelliert. Diese beiden Bedingungen lassen sich über den „Pass Criteria“ und „Composite Expression“ mit „1 OR 2“, wie in Abbildung 33 abgebildet, modellieren.

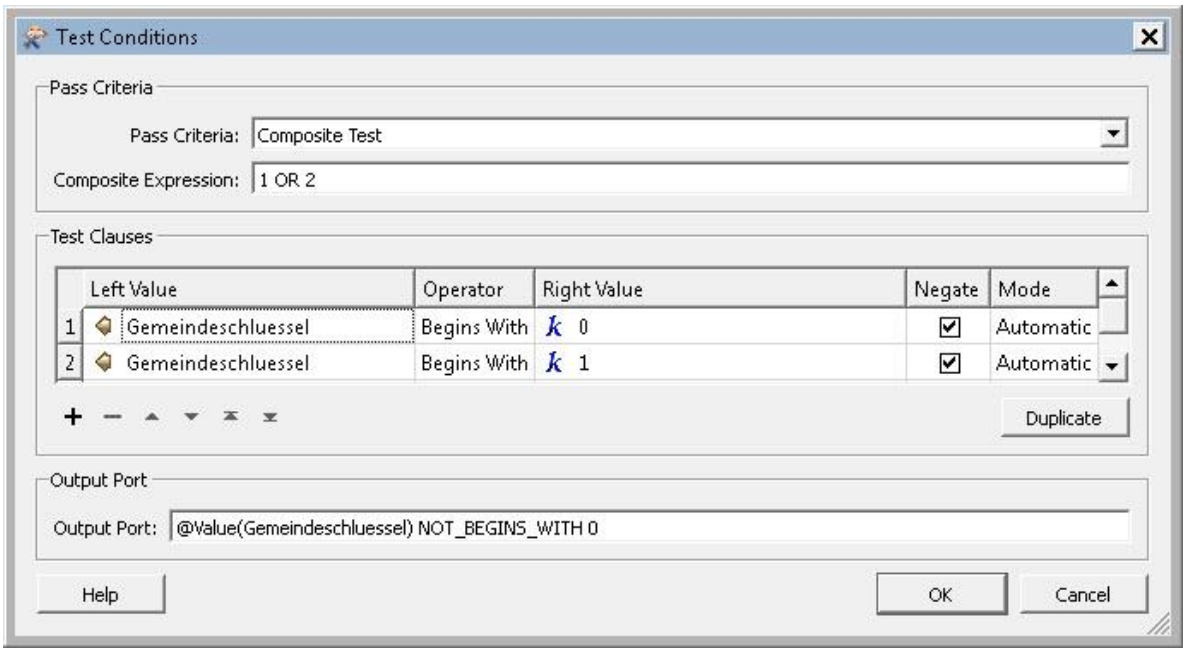

Abbildung 33: Prüfung des Gemeindeschlüssels in FME

Die Umsetzung des Prüfplans in OCL ist der zentrale Kernpunkt der vorliegenden Arbeit. Über die Prüfung des Attributes des amtlichen Gemeindeschlüssels hinaus können weitere Attribute mit Hilfe der FME überprüft werden, was jedoch nicht in dieser Arbeit weiter untersucht wurde.

Anhang B gibt einen Überblick über die Umsetzbarkeit der Prüfregeln in OCL und FME.

Nur ein Teil des FME-Workspaces kann schlussendlich mit Hilfe der genannten Methoden dynamisch umgesetzt werden. In OCL ist nur ein Teil der Prüfregeln realisierbar, in FME aber weitaus mehr.

Als Zwischenfazit lässt sich festhalten, dass mit Hilfe der Operatoren in FME im Vergleich zu den Operatoren in OCL wesentlich mehr Prüfregeln des AdV-Prüfplans umsetzen lassen, da FME eine wesentlich genauere Überprüfung der Attributwerte zulässt. Beispiele hierfür sind die Operatoren *contains* und *begins with*, die eine inhaltliche Überprüfung der Werte vornehmen und sich somit zusammengesetzte Attributwerte zum Beispiel aus Zahlen und Buchstaben (Beispiel: gml:id) validieren lassen.

Neben den Attributnamen und Attributwerten wurde in Abschnitt 4.4.3 der Frage nach der Umsetzung eines Operators nachgegangen. Dies ist jedoch nicht ohne weiteres möglich. In Tabelle 8 sind die Operatoren aus FME und OCL gegenübergestellt. Für einen Teil der Operatoren in FME gibt es kein Pendant in OCL. Für die Prüfung auf Existenz sind die Operatoren in FME differenzierter definiert als in OCL. Für diese Arbeit wurden die Operatoren in FME als gegeben angesehen und lediglich Attributwerte aus Codelisten dynamisch übersetzt.

Operator in OCL	**Operator in FME**
value = …	=
z. Bsp.: notEmpty()	!=
< …	<
> …	>
<= …	<=
>= …	>=
-	In
-	Between

-	Like
-	Matches Regex
-	Contains
-	Begins With
-	Ends With
isEmpty()	Attribute is Null
isEmpty()	Attribute is Empty String
isEmpty()	Attribute is Missing

Tabelle 8: Operatoren in FME und OCL (eigene Darstellung)

In einem kurzen Abriss wurde der Frage nachgegangen, welche Operatoren, die in den OCL-Konstrukten vorkommen, letztendlich auch in FME ausgedrückt werden können.
Die Überführung von OCL-Konstrukten in die interne Sprache von FME ist nicht problemlos zu realisieren. Der Grund ist die Umsetzung der Operatoren, die sich bei OCL und FME unterscheiden.

5 Prototypische Implementierung

Im vorherigen Kapitel wurde das Konzept erarbeitet, wie die Anforderungen der AdV in Form eines Prüfplanes in das Datenmodell implementiert werden können. Dabei wurde auf die Umsetzung der Attributnamen und Attributwerte eingegangen. Es wurde zwischen der Umsetzung von OCL nach FME und der Modellierung in FME unterschieden. In diesem Kapitel wird gezeigt, wie die Attribute und Attributwerte mit Hilfe von Python in FME umgesetzt werden. Es wird weiterhin aufgezeigt, welche weiteren Attribute manuell in dem FME-Workspace als Transformer angelegt werden mussten. Python überführt die Attributwerte aus OCL nach FME und wirkt in dieser Anwendung als Transformationssoftware. Letztendlich ist die Wahl der Programmiersprache nicht relevant. Python wurde für die Arbeit verwendet, da mit Python vergleichsweise schnell ein Programm mit grafischer Oberfläche geschrieben werden, Python XMI schnell parsen kann und es eine Anbindung an FME gibt, um den FME-Workspace nach der erfolgreichen Erstellung aus der Programmoberfläche starten zu können. Aus Zeitgründen konnte dies jedoch nicht mehr realisiert werden.

Im dritten Kapitel dieser Arbeit wurden bereits die notwendigen Softwarewerkzeuge näher erläutert. Die eingesetzten Technologien werden in diesem Kapitel nochmals aufgegriffen, wenn es um die Realisierung der Prüfsoftware geht.
Der Aufbau eines FME-Workspaces wurde im dritten Kapitel näher erklärt. In diesem Kapitel soll darauf eingegangen werden, wie die Umsetzung der Attributwerte durch die OCL-Konstrukte in die Struktur eines FME-Workspaces vorgenommen wird.
Im vorherigen Kapitel ist die Umsetzung der OCL-Konstrukte in die interne Sprache von FME behandelt worden. In diesem Kapitel soll das Konzept nun prototypisch, wie in Abbildung 34 zu sehen, umgesetzt werden.

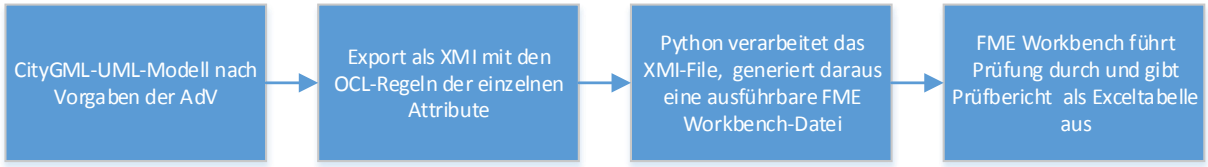

Abbildung 34: Workflow des Prototyps

5.1 Aufbau des FME-Workspaces

Ein FME-Workspace lässt sich auf unterschiedliche Arten erstellen. So kann man diese mit Notepad oder einem beliebigen Editor erstellen und bearbeiten (Safe Software Inc. 2009a).
Für den Aufbau eines dynamischen FME-Workspaces wurde der modellierte FME-Workspace mit dem Programm Notepad++ geöffnet, um herauszufinden, welche Teile statische Programmteile sind und welche Teile dynamisch erzeugt werden müssen. Aus dem statischen Teil des Workspaces wurde eine Vorlagendatei aufgebaut, welche durch den dynamisch erzeugten Teil mit Hilfe von Python komplettiert wird und letztendlich wieder einen vollständigen, ausführbaren FME-Workspace schreibt.
Die Feature Manipulation Engine ist das zentrale Werkzeug in dieser Arbeit. Zu den verwendeten Transformern gehören AttributeCreator, AttributeFilter, AttributeKeeper, FeatureMerger, Tester und TestFilter.
AttributeCreator: Fügt dem vorhandenen Feature weitere Attribute hinzu. Diesen Attributen müssen konstante Werte zugewiesen werden.

AttributeFilter: Übergibt Features an unterschiedliche Ausgänge, abhängig von ihren Werten. Mögliche Ausgänge können manuell gesetzt werden oder von Inputdatensätzen extrahiert werden.
AttributeKeeper: Wenn für die weitere Verwendung nur gewisse Attribute zur Verfügung stehen sollen, wird der AttributeKeeper verwendet. Das Pendant zum AttributeKeeper ist der AttributeRemover.
FeatureMerger: Der Transformer überträgt die Attribute oder die Geometrie von einem Feature auf ein anderes. Wenn mehrere Features vorhanden sind, lässt sich auswählen, welches Feature für das Zusammenfügen verwendet werden soll. Der Transformer verfügt über zwei Eingänge, den Requestor und den Supplier. Features, die die gewünschten Attribute aufweisen, werden über den Supplier verbunden und Features, die die Attribute erhalten, werden über den Requestor verbunden. Die Features werden über die ID miteinander verbunden. In den Einstellungen lässt sich zudem festlegen, dass auch die Werte des Suppliers weiterverwendet werden können. Darüber hinaus lässt sich einstellen, dass nur die Attribute erhalten bleiben sollen.
Tester: Der Tester filtert Features nach deren Werten und gibt diese als „passed“, also bestanden oder als „failed“ – nicht bestanden aus. Dieser Transformer wird verwendet, wenn lediglich nach „Attribute Is Empty String“ oder „Attribute Is Missing“ gefiltert wird oder werden kann.
TestFilter: Der Testfilter erweitert die Funktionsweise des Testers um die Möglichkeit, neben den oben genannten Kriterien auch nach weiteren Werten zu filtern. Dieser Transformer wird verwendet, wenn die Werte über Codelisten zuordenbar sind und nicht leer sein dürfen. Auf die Prüfung auf Existenz eines Attributwertes wurde bereits im vorherigen Kapitel ausführlich eingegangen.
Excel-Writer: Im Falle eines CityGML-Datensatzes im Detaillierungsgrad LOD1, so ist das Ergebnis eine eindimensionale Excel-Tabelle, bestehend aus dem Feature Type *Building*. Darin werden fehlerhafte Datensätze aufgeführt.
Handelt es sich um einen CityGML-Datensatz im Detaillierungsgrad LOD2 ist das Ergebnis eine mehrdimensionale Excel-Tabelle. In einem Tabellenblatt werden die fehlerhaften Datensätze mit den Attributen des Features Typs *Building* und in einem zweiten Tabellenblatt

die fehlerhaften Datensätze mit den Attributen des Features Typs für *BuildingPart* dargestellt.

Wenn ein *BuildingPart* eines CityGML-Datensatzes fehlerhaft ist, erscheint lediglich das *BuildingPart* in dem Excel-Tabellenblatt, jedoch wird nicht automatisch das *Building* als fehlerhaft in das Tabellenblatt *Building* geschrieben. Diese beiden Teile werden somit getrennt betrachtet, aber über die „gml:id“ mit den jeweils fehlerhaften Attributwerten aufgeführt. Dadurch liegen Attributnamen und Werte aus der Codeliste in der erzeugten FME-Workbench-Datei vor. Diese kann ausgeführt werden, um die Attribute und die Attributwerte zu überprüfen.

Der FME-Workspace besteht aus den genannten Standard-Transformern (Testfilter, AttributeCreator, etc.) und wurde zuvor für die zu füllende Vorlage in Python erstellt. Es ist jeweils ein Transformer-Konstrukt aus Tester oder Testfilter, AttributeCreator und AttributeKeeper für ein zu prüfendes Attribut vorhanden.

Ein Auszug aus dem erstellten FME-Workspace für die Prüfung von CityGML-LOD2-Daten-sätzen ist in Abbildung 35 zu sehen. Die für die Attributprüfung notwendigen Transformer stellen einen Teil des FME-Workspaces dar, während ein Großteil für den Aufbau der grafischen Oberfläche und Kommentare verwendet wird.

```
4713  XLSXW2_1_DEF BuildingPart   xlsx_drop_sheet              No   xlsx_tr
4714  # ------------------------------------------------------------------
4715  XLSXW2_1_DEF Building   xlsx_drop_sheet             No   xlsx_trunc_
4716  # ------------------------------------------------------------------
4717  FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_citygml_function_Te
4718  FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_3 ATTRSET_
4719  FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_citygml_roof_type_T
4720  FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_9 ATTRSET_
4721  FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleLage_Tes
4722  FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_15 ATTRSET
4723  FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleBodenhoe
4724  FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_11 ATTRSET
4725  FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleDachhoeh
4726  FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_13 ATTRSET
```

Abbildung 35: Dynamischer Teil des erstellten FME-Workspaces umrandet (LOD2)

Ein beispielhafter Auszug einer kompletten Zeile für den Attributnamen „DatenquelleLage“ und den zugehörigen Attributwerten ist wie folgt abgebildet:

FACTORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_10 ATTRSET_CREATE_DIRECTIVES _PROPAGATE_MISSING_FDIV INPUT FEATURE_TYPE TestFilter_DatenquelleLage_<at>Value<openparen>DatenquelleLage<closeparen><space>NOT_IN<space>1000<comma>1100<comma>1200<comma>1300<comma>2000<comma>3000 ATTR DatenquelleLage VALUE_IS_INCORRECT OUTPUT OUTPUT FEATURE_TYPE AttributeCreator_10_OUTPUT

In den Abbildungen 36 und 37 sind die dynamischen Teile des fertigen FME-Workspaces umrandet. Hiervon sind wiederum ein Teil dynamisch über die Codelistenwerte befüllt.

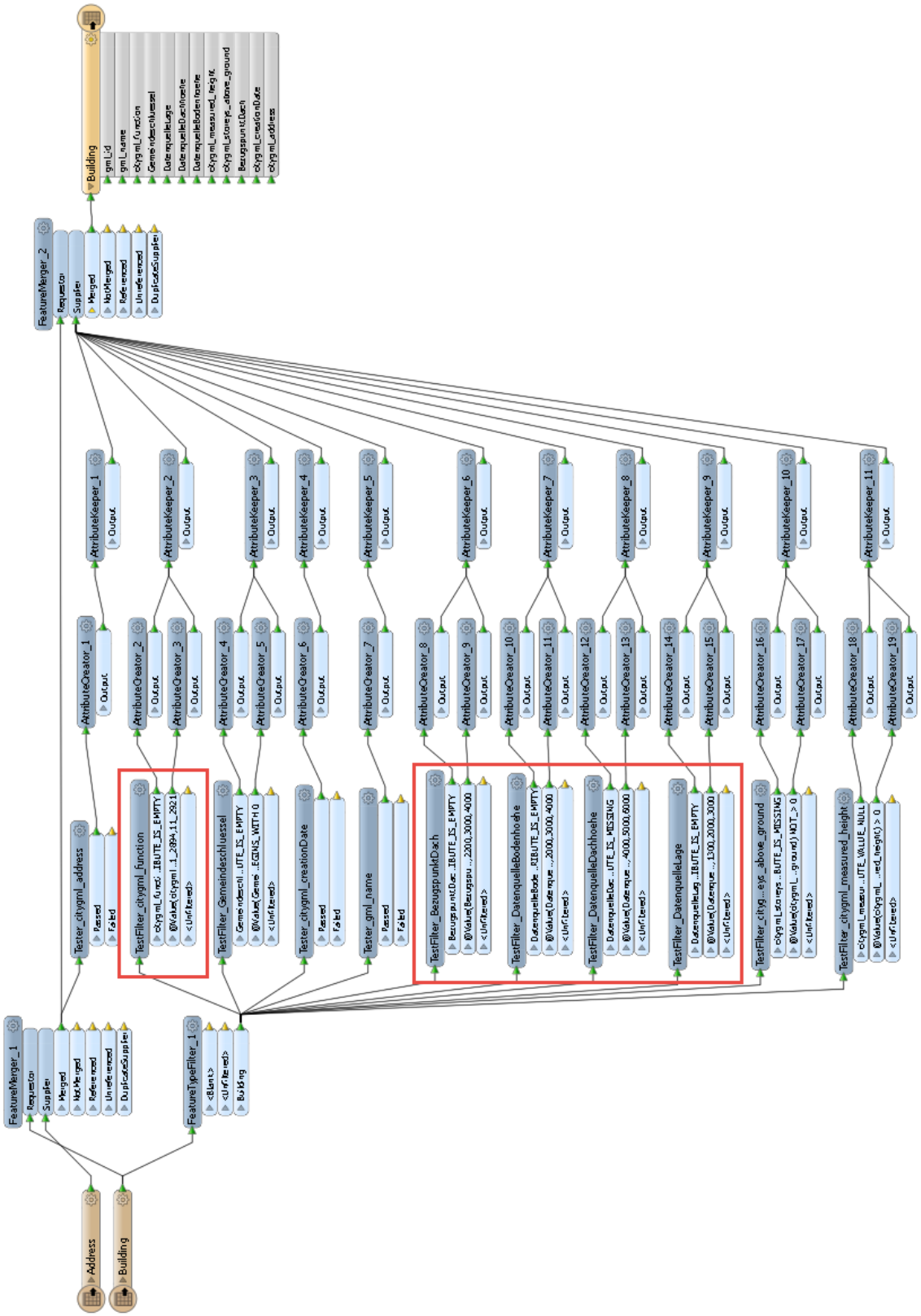

Abbildung 36: Umrandet der dynamische Teil des FME-Workspaces (LOD1)

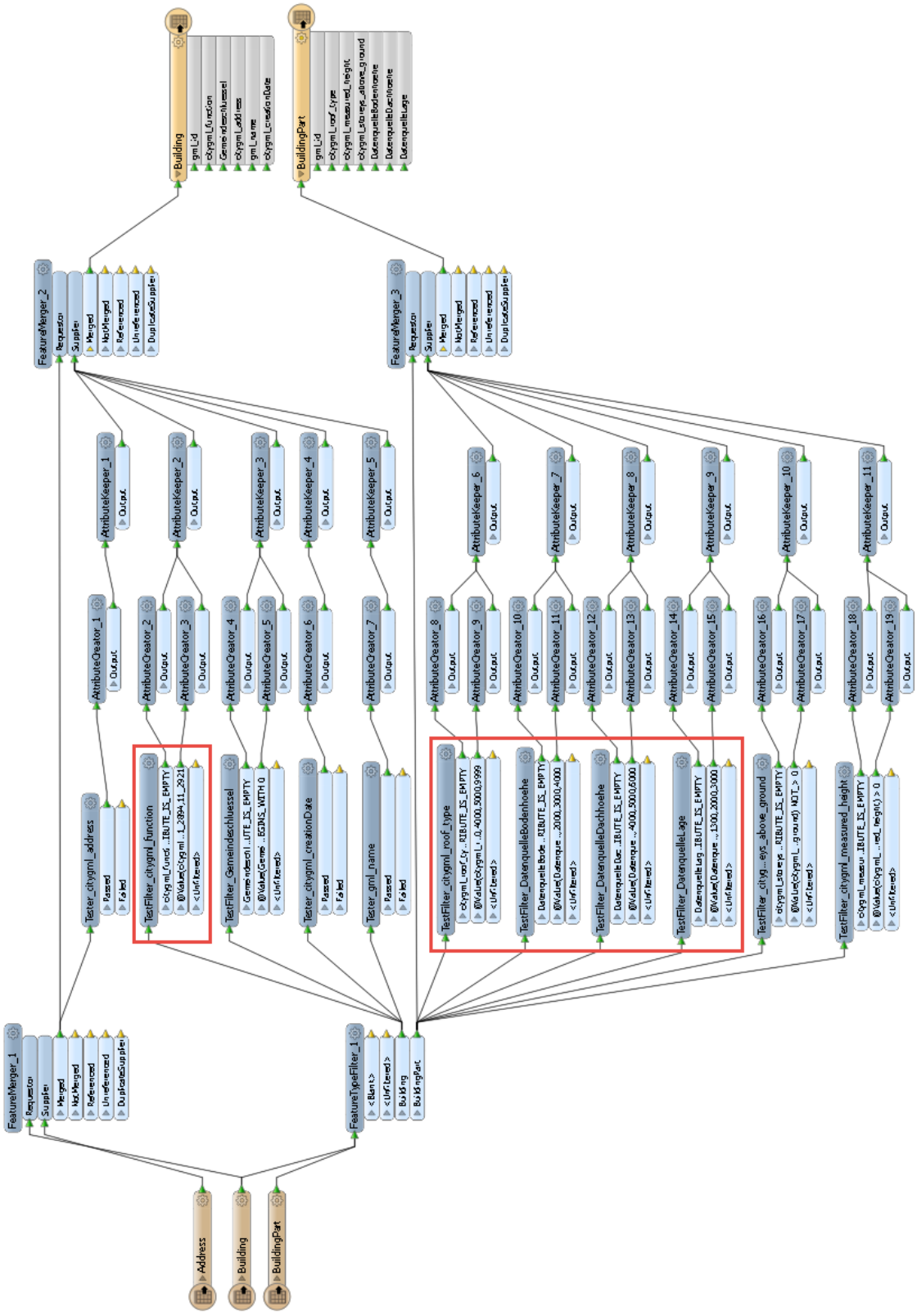

Abbildung 37: Umrandet der dynamische Teil des FME-Workspaces (LOD2)

5.2 Vorlagenkonzept

Das Vorlagenkonzept ist ein wesentlicher Teil der Prüfsoftware. Die Auswahl der richtigen Vorlage ist abhängig vom Detaillierungsgrad der zu prüfenden CityGML-Datensätze. Hier muss bei den Vorlagen zwischen LOD1 und LOD2 unterschieden werden. Die zentrale Vorlage für die Prüfung ist die FME-Vorlage. Diese Datei stellt den statischen Teil eines FME-Workspaces dar und beinhaltet einen Großteil der Überprüfung der Attributwerte, welche in der FME-Workbench modelliert wurden. Diese Vorlage wird durch den Konvertierungsprozess, den dynamischen Teil der Arbeit mit den Attributen und Attributwerten „vervollständigt“. Diese ist anschließend ausführbar und stellt eine valide FME-Workbench-Datei dar. In Abschnitt 3.3.1 wurde genauer auf den Aufbau eines FME-Workspace eingegangen. Im Kapitel zum Spatial-ETL-Werkzeug FME wurde auf den Aufbau eines FME-Workspaces eingegangen. Neben den in der Benutzeroberfläche sichtbaren Readern, Transformern und Writern werden in der FME-Workbench-Datei auch Informationen zur Lage der drei genannten Komponenten, zu Pfaden der Eingabe und Ausgabedateien, zum Pfad des FME-Workspaces, sowie Kommentare in Form von Lesezeichen mit abgespeichert. Abbildung 34 veranschaulicht das Verhältnis des statischen zum dynamischen Teil (umrandet) eines FME-Workspaces. Von 4726 Zeilen in dem FME-Workspace sind 10 Zeilen dynamischer Code, welcher von Python erzeugt wird. Dies spricht für die Verwendung bereits halbfertiger Vorlagen.
Neben der FME-Vorlage gibt es noch die Excel-Vorlage. Diese Vorlage ist ein leeres Excel-Dokument, welches bereits im Ausgabeordner vorliegen muss. In diese Vorlage wird bei der Ausführung des Prüfwerkzeuges der Prüfbericht geschrieben. Der Grund, weshalb für den Prüfbericht ein Dateiname nicht dynamisch vergeben werden kann, ist der, dass es derzeit nicht möglich ist, diesen Pfad in der FME-Vorlage zu ändern. Dies konnte im Rahmen der Arbeit nicht mehr umgesetzt werden.

5.3 Programmoberfläche

Mit Hilfe der Programmiersprache Python wird die grafische Oberfläche des Prüftools erstellt, die exportierte XMI geparst und mit den „extrahierten“ Attributen und zugehörigen Attributwerten die Vorlage gefüllt.

Die Programmoberfläche des Prüftools ist in zwei Komponenten unterteilt. In der ersten Komponente werden die benötigten Dateien geladen, wobei hier zwischen LOD1 und LOD2 unterschieden wird. Dies liegt daran, dass unterschiedliche Attribute bei den beiden Detaillierungsstufen vorliegen und diese auf unterschiedlichen Gebäudeteilen verteilt sind.
Der Anwender muss folglich vor Prüfung eines Datensatzes bereits wissen, welchen Detaillierungsgrad seine zu prüfenden CityGML-Daten aufweisen. In Abbildung 38 ist die Programmoberfläche der Prüfsoftware veranschaulicht.
Dies spielt eine Rolle für die Felder „XMI-Datei auswählen“, „FME-Vorlage auswählen“ und „CityGML-Datensatz auswählen“. Im Anhang C, in den Abschnitten des Programmcodes ist dieser Teil in Abschnitt 2 zu finden. Das Prüfungstool ist übersichtlich und einfach aufgebaut. Es soll die Möglichkeit bieten, notwendige zu verwendende Dateien einzubinden. Dabei ist das Wichtigste die in Enterprise Architect erzeugte XMI-Datei mit dem CityGML-Modell und den Qualitätsangaben nach der AdV. Diese beinhaltet alle notwendigen Attribute und Attributwerte, die aus den Codelisten stammen.
Der zweite Teil des Softwaretools behandelt die Ausgabedateien. Hier werden die Namen und Pfade der Ausgabedateien angegeben. Die Ausgabedateien sind der FME-Workspace, welche unter „CityGML-Prüfung Name anlegen“ und der Prüfbericht, dessen Pfad unter „Prüfbericht Pfad anlegen“ und der Name unter „Prüfbericht Name anlegen“ angelegt bzw. eingetragen werden müssen.
Wenn die entsprechenden Vorlagen ausgewählt, Pfade angelegt und Namen vergeben werden, wird im ersten Schritt die Konvertierung der XMI-Datei vorgenommen. Nach dem Konvertierungsprozess liegen die Attribute und zugehörigen Werte im temporären Speicher vor. Es wurde dabei ein Dictionary aufgebaut, um sämtliche Werte eines Attributes einzeln abzufragen und in die Struktur für den jeweiligen Transformer zu bringen. Darauf wird im nächsten Abschnitt näher eingegangen.
Im zweiten Schritt, der über den Button „Konvertierung starten“ ausgeführt wird, werden die Vorlagendateien mit diesen Attributen und Werten gefüllt und ein ausführbarer FME-Workspace erzeugt. Nach dem manuellen ausführen dieses FME-Workspaces liegt der Prüfbericht anschließend im Ausgabeordner bereit.

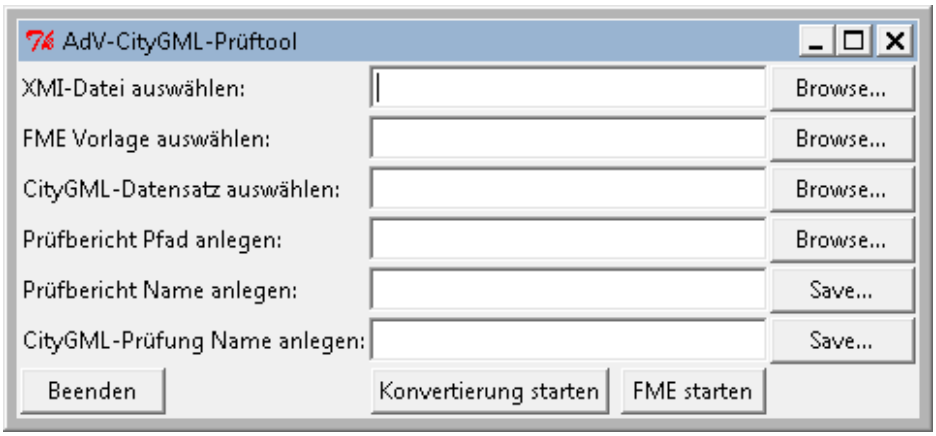

Abbildung 38: Python Programmoberfläche

5.4 Konvertierungsprozess

Der Konvertierungsprozess wird manuell angestoßen und die XMI-Datei mit den generischen Attributen und den Attributwerten in den temporären Speicher geladen. Im zweiten Schritt wird die Vorlage gefüllt und die FME-Workbench-Datei geschrieben. Dieser mehrfache Konvertierungsprozess soll in diesem Abschnitt näher erklärt werden. Das AdV-CityGML-Modell aus Enterprise Architect, welches als XMI-Datei exportiert wurde, wird beim Starten des Konvertierungsprozesses in die Bestandteile Attributname und Attributwerte zerlegt.

Dabei werden die Attribute und Attributwerte in einem temporär vorliegenden Dictionary gespeichert und liegen für den weiteren Konvertierungsprozess vor. Dieser Vorgang wird generell als parsen (englisch für analysieren oder grammatikalisch zergliedern) bezeichnet. Programmtechnisch wird das XML-Dokument durch ein komplexes Aggregat verschiedener Objekte dargestellt (Behnel 2015).

XMI Dateien können einerseits auf „Wohlgeformtheit“ und andererseits auf „Validität“ hin überprüft werden. In unserem Fall soll der Inhalt der XMI-Datei in eine FME-Workbench-Datei überführt werden, weshalb die Überprüfung auf „Wohlgeformtheit“ und „Validität“ nicht nötig ist (Mertz 2003). Aufgrund des XML-Formats können entsprechende Dateien leicht erzeugt, durchsucht, weiterverarbeitet, gespeichert und über das Internet übertragen werden.

Damit die Attribute und deren Werte für die weitere Verarbeitung zur Verfügung stehen, werden diese, wie bereits erwähnt, in einem Dictionary gespeichert. Ein Dictionary enthält beliebig viele Schlüssel-Wert-Paare (key-value pairs), wobei der Schlüssel nicht unbedingt, wie bei einer Liste, eine ganze Zahl sein muss. Der Datentyp dict ist mutable (englisch: veränderlich) (Ernesti und Kaiser 2009).
Die OCL-Konstrukte innerhalb der XMI 2.4.1 sind wie folgt abgelegt:

```
body="inv: name = 'DatenquelleBodenhoehe'  and (value = 1000 or value =
1100 or value = 1200 or value = 1300 or value = 1400 or value = 1500 or
value = 1600 or value = 1700 or value = 1800 or value = 2000 or value =
3000 or value = 4000)"/>

body="inv: name = 'Gemeindeschluessel' and value -&amp;gt;notEmpty()
```

In FME sieht das für das Attribut „DatenquelleBodenhoehe“ vorcodiert wie folgt aus:

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleBodenhoehe_TestFactory_0 INPUT FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERINPUTLINE_0 TEST DatenquelleBodenhoehe ATTRIBUTE_IS_EMPTY "" ENCODED TEST DatenquelleBodenhoehe ATTRIBUTE_IS_MISSING "" ENCODED BOOLEAN_OPERATOR COMPOSITE COMPOSITE_TEST_EXPR "1 OR 2" OUTPUT PASSED FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_DatenquelleBodenhoehe<space>ATTRIBUTE_VALUE_NULL OUTPUT FAILED FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERINPUTLINE_1

In FME sieht das für das Attribut „Gemeindeschluessel“ vorcodiert wie folgt aus:

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_Gemeindeschluessel_TestFactory_0 INPUT FEATURE_TYPE TestFilter_Gemeindeschluessel_TESTFILTERINPUTLINE_0 TEST Gemeindeschluessel ATTRIBUTE_IS_EMPTY "" ENCODED TEST Gemeindeschluessel ATTRIBUTE_IS_MISSING "" ENCODED BOOLEAN_OPERATOR COMPOSITE COMPOSITE_TEST_EXPR "1 OR 2" OUTPUT PASSED FEATURE_TYPE TestFilter_Gemeindeschluessel_Gemeindeschluessel<space>ATTRIBUTE_VALUE_NULL OUTPUT FAILED FEATURE_TYPE TestFilter_Gemeindeschluessel_TESTFILTERINPUTLINE_1

Die OCL-Konstrukte der gültigen Wertebereiche sind als Text abgelegt und nicht weiter voneinander getrennt, so dass man die einzelnen Werte herauslesen kann.
Für den Aufbau des Dictionary werden die Werte von links nach rechts der OCL-Constraints geparst und abgelegt. Diese OCL-Konstrukte müssen eine bestimmte Form aufweisen, um von Python in einem Dictionary abgelegt werden zu können. Trennzeichen ist hier einerseits das “or”, um die Werte zu unterscheiden und das “=” um zwischen “value” und dem tatsächlichen Wert zu unterscheiden. Die Attribute und Werte werden jetzt jeweils in eine Zeile geschrieben. In diesem Fall wird zu den OCL-Konstrukten innerhalb der XMI wie gefolgt geparst:
Pythoncode:

```
# Parsen zu den Constraints der Attribute
for con in cons_in.xpath("//specification"):
        for cons in con.xpath("./@body"):
        # Gibt die OCL-Konstrukte zu allen Attributen aus
        print cons
```

Der zugehörige Programmcode ist in Anhang C in Abschnitt 3 abgelegt. Im Dictionary werden die Attributwerte, als auch die Werte, welche nicht Null sein dürfen und welche größer Null sein müssen temporär gespeichert. Diese sind in Abbildung 39 zu sehen.

```
DatenquelleDachhoehe
['1000', '2000', '3000', '4000', '5000', '6000']
gml:name
notEmpty()
Gemeindeschluessel
notEmpty()
storeysAboveGround
['and value &gt;']
DatenquelleLage
['1000', '1100', '1200', '1300', '2000', '3000']
citygml_roof_type
['1000', '2100', '2200', '3100', '3200', '3300', '3400', '3500', '3600', '3700', '3800', '3900',
address
notEmpty()
measuredHeight
['and value &gt;']
externalReference
notEmpty()
```

Abbildung 39: Auszug aus dem Dictionary

Im Abschnitt 4 des Anhangs C ist der Programmcode zur Erstellung des Dictionaries zu finden. Im darauffolgenden Abschnitt 5 des Programmcodes im Anhang C werden lediglich die Attribute, welche mit Wertelisten verknüpft bleiben im Dictionary gespeichert. Die anderen werden gelöscht. Im nächsten Schritt werden diese Attributnamen und Attributwerte in eine Vorlagenstruktur des FME-Workspaces verarbeitet und als fertiger FME-Workspace geschrieben. Dieser Teil stellt den dynamischen Teil der schlussendlich fertigen und ausführbaren FME-Workbench-Datei dar. Der folgende Codeauszug (Anhang C Abschnitt 6) des Python-Codes veranschaulicht dies:

```
zeile1 = 'FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_' + str(key)
+ '_TestFactory_' + str(i) + ' INPUT FEATURE_TYPE TestFilter_' + str(key)
+ '_TESTFILTERINPUTLINE_' + str(i) + ' TEST @EvaluateExpres-
sion(FDIV,STRING_ENCODED,<at>Value<openparen>' + str(key) +
'<closeparen>,TestFilter_' + str(key) + ') NOT_IN '

werte = ''

for valuex in value:

      werte = werte + valuex + '<comma>'

werte = werte[0:len(werte)-7] + ' ENCODED BOOLEAN_OPERATOR OR COMPO-
SITE_TEST_EXPR <Unused> OUTPUT PASSED FEATURE_TYPE TestFilter_' +
str(key) + '_<at>Value<openparen>' + str(key) + '<closeparen><space>'
```

In diesem Programmcode werden Attributnamen aus dem Dictionary (key) und Attributwerte (value) ausgelesen und Zeile für Zeile in den hardcodierten Teil der FME-Factories, bestehend aus zwei unterschiedlichen Variablen, jeweils für Attributnamen und Attributwerte geschrieben. Das Ergebnis schaut wie folgt aus:

FACTORY_DEF * TestFactory FACTORY_NAME TestFilter_DatenquelleBodenhoehe_TestFactory_1 INPUT FEATURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERINPUTLINE_1 TEST @EvaluateExpression(FDIV,STRING_ENCODED,<at>Value<openparen>DatenquelleBodenhoehe<closeparen>,TestFilter_DatenquelleBodenhoehe) NOT_IN 1000<comma>1100<comma>1200<comma>1300<comma>1400<comma>1500<comma>1600<comma>1700<comma>1800<comma>2000<comma>3000<comma>4000 ENCODED BOOLEAN_OPERATOR OR COMPOSITE_TEST_EXPR <Unused> OUTPUT PASSED FEA-

TURE_TYPE TestFilter_DatenquelleBodenhoehe_<at>Value<openparen>DatenquelleBoden-
hoehe<closeparen><space>NOT_IN<space>
1000<comma>1100<comma>1200<comma>1300<comma>1400<comma>1500<comma>1600<co
mma>1700<comma>1800<comma>2000<comma>3000<comma>4000 OUTPUT FAILED FEA-
TURE_TYPE TestFilter_DatenquelleBodenhoehe_TESTFILTERINPUTLINE_2

Die erstellte FME-Workbench-Datei muss manuell ausgeführt werden. Das Ergebnis des Prüfprozesses ist beispielhaft in der Abbildung 40 zu sehen. Das Prüfprotokoll im Excel-Format zeigt fehlerhafte CityGML-Daten geordnet nach „gml_id.“

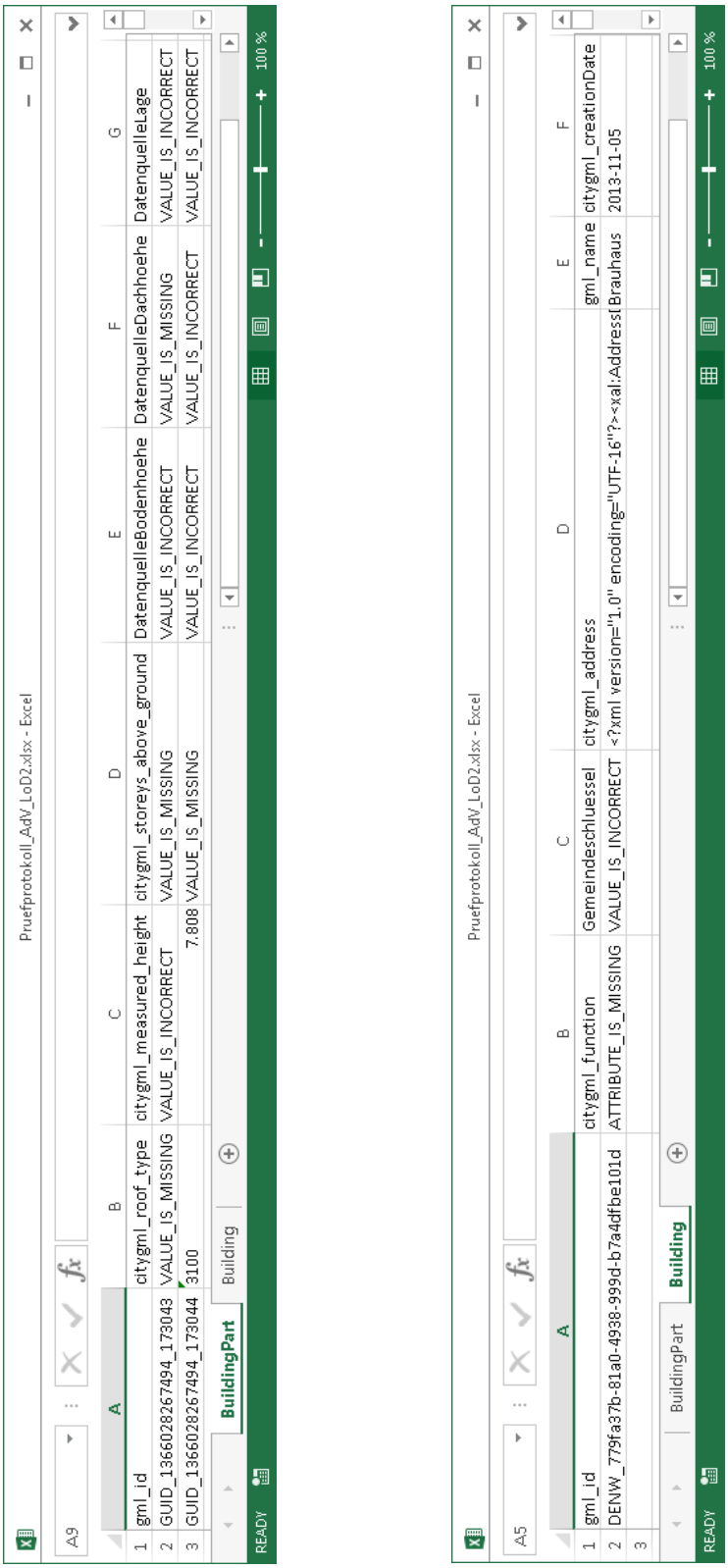

Abbildung 40: Prüfprotokoll mit fehlerhaften Datensätzen (LOD2) in Excel

Der Konvertierungsprozess lässt sich somit auf folgende 3 Teile beschränken.

1. Laden der Eingabe- und Ausgabedatensätze bzw. Anlegen deren Pfade.
2. Starten der Auslesung der XMI-Datei (LOD1 oder LOD2) und Füllung des temporären Speichers in Form eines Dictionaries mit Attributnamen und –werten.
3. Konvertierung der Attributnamen und -werte in die Vorlagenstruktur und Herausschreiben einer fertigen FME-Workbench-Datei zur Prüfung des LOD1 oder LOD2 Datensatzes.
4. Manuelles Starten der FME-Workbench zur Ausführung der Prüfung und Erstellung des Prüfprotokolls.

Aus dem Programm heraus soll zukünftig FME automatisch gestartet und der erzeugte Workspace ausgeführt werden. Dies konnte im Rahmen dieser Arbeit nicht mehr realisiert werden. Python erzeugt einen Teil des FME-Workspaces hardcodiert und überführt letztlich nur die Attributwerte der Attribute *rooftype*, *functiontype* und der generischen Attribute im dynamischen Teil in die Verarbeitung eines ausführbaren FME-Workspaces.
In dieser Arbeit wurden lediglich die Attributnamen und Attributwerte aus dem UML-Modell dynamisch in eine FME-Workbench-Datei umgesetzt. Wenn zukünftige FME-Workspaces in der Modellierungssoftware Enterprise Architect modelliert werden sollen und nur noch per Transformationssoftware in einen ausführbaren FME-Workspace überführt werden soll, dann kann der Ansatz, der in dieser Arbeit verfolgt wurde einen Beitrag dazu leisten, die interne Sprache von FME und die Konvertierungsprozesse zu verstehen. Ziel ist es schließlich, neben den Attributwerten auch automatisiert und dynamisch die Operatoren aus der Modellebene in die Datenebene zu überführen.

6 Evaluierung des Prototyps

In dem folgenden Kapitel soll die in dieser Arbeit entwickelte prototypische Anwendung hinsichtlich der Korrektheit geprüft werden. Durch CityGML-Datensätze, die dem AdV-Profil entsprechen sollen, steht für die Prüfung ein geeigneter Anwendungsfall bereit. In dem Hauptteil der Evaluierung werden die zur Verfügung stehenden CityGML-Testdatensätze vorgestellt, bevor anschließend diese anhand dreier Testfälle durchgeführt wird.

6.1 Anwendungsfall CityGML-Daten nach AdV-Profil

6.1.1 Beschreibung des Anwendungsfalls

Der Anwendungsfall umfasst die Prüfung der Vorgaben der AdV für 3D-Stadtmodelle, welche in Form eines Prüfplans teilweise als OCL-Konstrukte in das UML-Modell verarbeitet und anschließend in eine Prüfsoftware überführt wurden. Dies ist im Kapitel 4 ausführlich beschreiben. Die CityGML-Datensätze, die diesen Vorgaben genügen müssen, sollen im nächsten Abschnitt hinsichtlich dieses überführten AdV-CityGML-Profils überprüft werden. Diese Testdaten sollen nun näher beschrieben werden.

6.1.2 Testdaten

Als Testdaten stehen CityGML-Datensätze der Detaillierungsstufen LOD1 und LOD2 zur Verfügung. Diese stammen aus Nordrhein-Westfalen. Die verwendeten Testdatensätze aus Thüringen und Bayern liegen lediglich im LOD1 vor. Alle Datensätze dienen der Überprüfung der korrekten Umsetzung des AdV-CityGML-Profils, als auch der Evaluierung der Prüfsoftware.

6.1.2.1 CityGML-Daten der AdV aus Nordrhein-Westfalen

Für die Evaluierung der Prüfsoftware werden Testdaten für Nordrhein-Westfalen verwendet, die durch die AdV zur Verfügung gestellt werden. Diese wurden von der Internetseite der

AdV bezogen. Die CityGML-Datensätze wurden unter: http://repository.gdi-de.org/schemas/adv/citygml/Beispieldaten/ heruntergeladen. Diese CityGML-Daten liegen in den Detaillierungsgraden LOD1 und LOD2 vor.

In Abbildung 41 und 42 sind die AdV-CityGML-Beispieldatensätze im LOD1 und LOD2 mit Hilfe des *FME Data Inspectors* visualisiert.

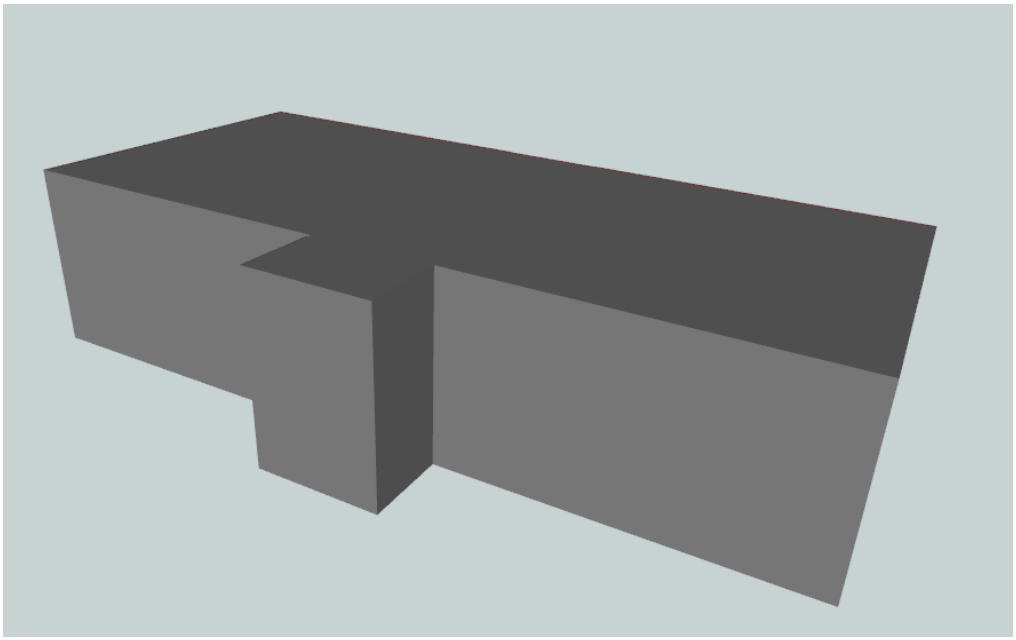

Abbildung 41: AdV-CityGML-Beispieldatensatz LOD1

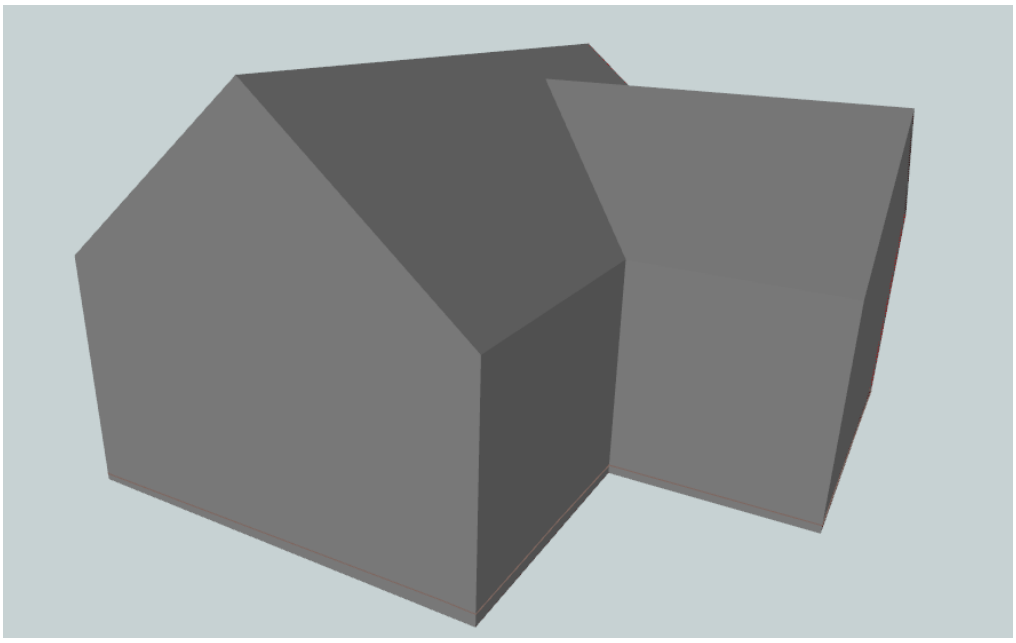

Abbildung 42: AdV-CityGML-Beispieldatensatz LOD2

6.1.2.2 CityGML-Daten des LDBV Bayern

Als zweiter Testfall für die Überprüfung von CityGML-Daten nach dem AdV-Profil werden CityGML-Daten des LDBV Bayern in dem Detaillierungsgrad LOD1 verwendet. Die Testdaten des LDBV wurden unter: http://www.ldbv.bayern.de/service/testdaten.html heruntergeladen.

Die CityGML-Datensätze des LDBV Bayern weisen, wie bereits in Kapitel 2 erwähnt, über das AdV-CityGML-Profil hinausgehend weitere Attributdaten auf. Diese wurden in dieser Arbeit nicht weiter untersucht und müssen bei Abgabe an das ZSHH aus den Datensätzen entfernt werden (AdV 2015d). Diese werden durch den Anwendungsfall „AdV-Profil“ nicht abgedeckt.

In Abbildung 43 ist der verwendete Beispieldatensatz im LOD1 des LDBV Bayern mit Hilfe des *FME Data Inspectors* visualisiert.

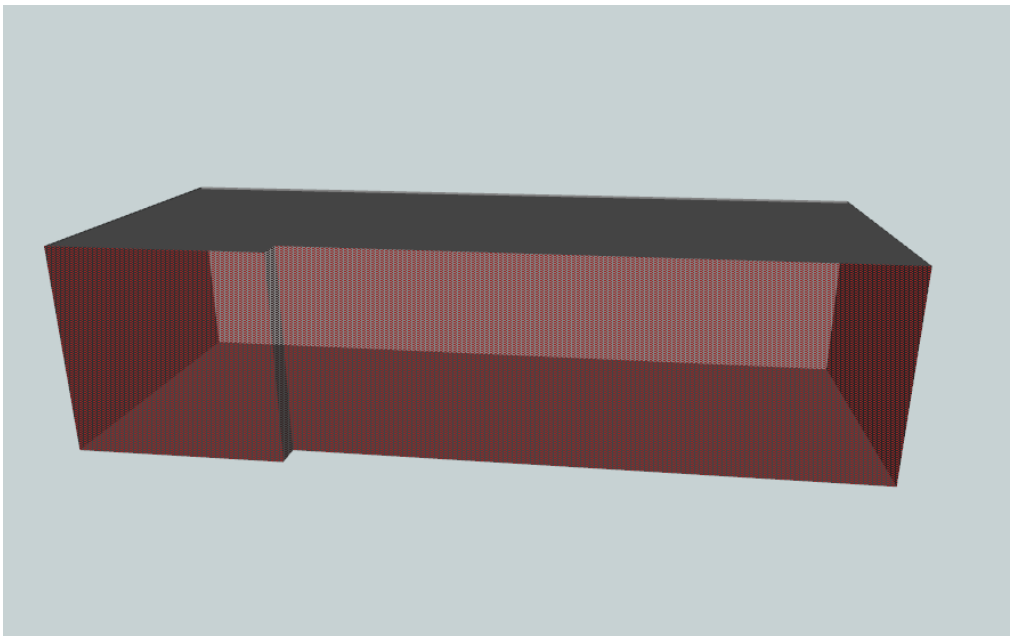

Abbildung 43: CityGML-Beispieldatensatz LOD1 des LDBV Bayern

6.1.2.3 CityGML-Daten der AdV aus Thüringen

Als dritten Testfall für die Überprüfung von CityGML-Daten nach dem AdV-Profil werden CityGML-Daten aus Thüringen in dem Detaillierungsgrad LOD1 verwendet. Die Testdaten aus Thüringen werden durch die AdV unter: http://www.adv-online.de/AdV-Produkte/Standards-und-Produktblaetter/ZSHH/ zur Verfügung gestellt.
In Abbildung 44 ist der verwendete Beispieldatensatz im LOD1 des Bundeslandes Thüringen mit Hilfe des *FME Data Inspectors* visualisiert.

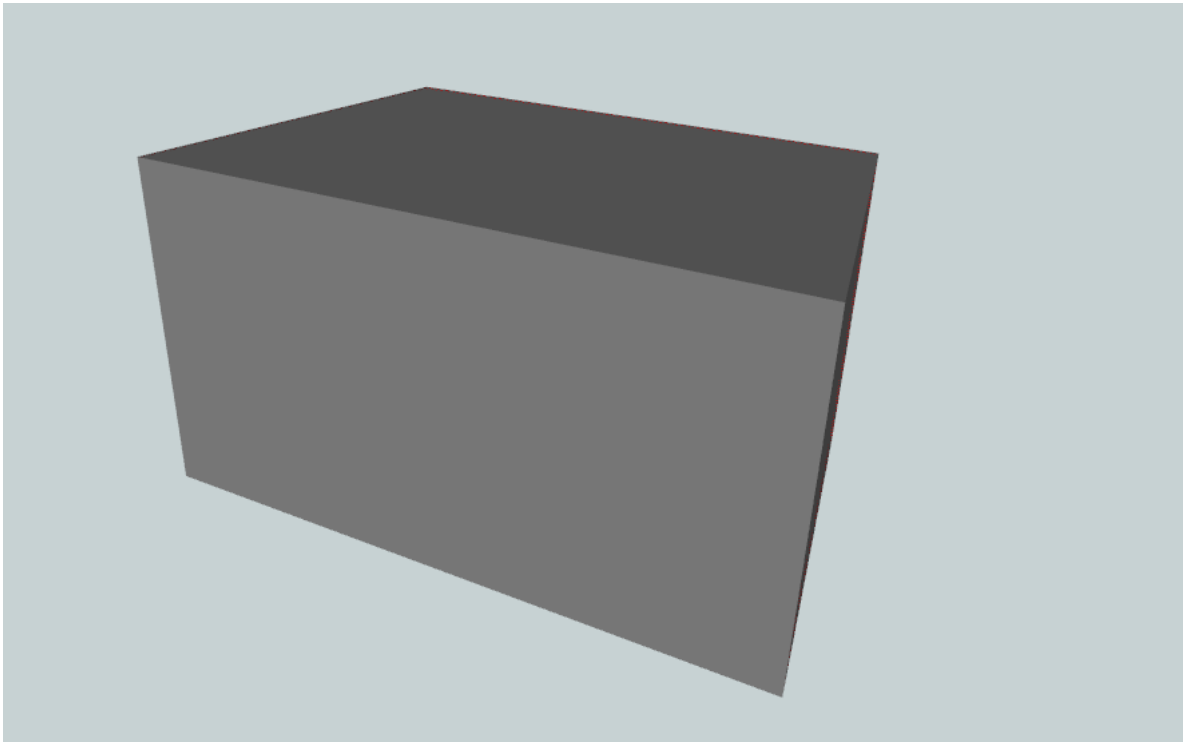

Abbildung 44: CityGML-Beispieldatensatz LOD1 aus Thüringen

6.2 Testdurchführung und Bewertung

Den bayerischen und thüringischen CityGML-Daten musste eine Adresse hinzugefügt werden. Hierfür wurde auf den Testdatensatz der AdV zurückgegriffen. Das externe Modul xal (Extensible Adress Language) wurde hierfür aus dem AdV-CityGML-Beispieldatensatz des LOD1 übernommen und in den Testdatensatz integriert. Dadurch ist eine Überprüfung des CityGML-Datensatzes möglich. In den AdV-CityGML-Datensätzen sind Adressen bereits als Verknüpfungselement geführt.

Für die Testdurchführung werden die CityGML-Daten der AdV als auch des LDBV Bayern und des Bundeslandes Thüringen mit Hilfe des Prüfprogramms hinsichtlich der Attributwerte geprüft. Daraufhin werden die fehlerhaften Datensätze als Excel-Prüfprotokoll ausgegeben. Der Prüfplan für Gebäudemodelle LOD1/LOD2 (AdV 2015d) listet die zu prüfenden Attribute mit den gültigen Wertebereichen auf.

6.2.1 Testfall 1

Die Prüfung der CityGML-Daten der AdV bezieht sich auf die Beispieldatensätze LOD1 und LOD2. Für die Erstellung der Prüfsoftware wurden die AdV-CityGML-Daten als Referenz verwendet, weshalb diese folglich auch keine fehlerhaften oder fehlenden Attributwerte enthalten.

Um die Prüfsoftware auf Korrektheit der Prüfergebnisse zu prüfen, wurden die Daten manipuliert. Dadurch müssen die manipulierten Testdaten im Prüfprotokoll enthalten sein. In Tabelle 9 sind die originalen und die manipulierten Attributwerte für den LOD1 aufgeführt.

Dabei wurden die Attributwerte des AdV-CityGML-Datensatzes entfernt oder die Attributwerte verändert. Der CityGML-Testdatensatz im LOD1 heißt im Original „Beispieldatensatz_ AdV-CityGML-Profil_LOD1.xml“. Der manipulierte CityGML-Datensatz trägt den Namen „AdV-CityGML-Profil_LOD1_manipulated.xml“.

gml:id=DENW_110e8edf-dda2-4130-a564-87b2a3cb3f35			
Building	Attributname	Attributwert original	Attributwert manipuliert
	function	1121	-
	gml_name	-	-
	Gemeindeschluessel	05562008	253456789
	DatenquelleDachhoehe	1000	2
	DatenquelleLage	1000	1235
	DatenquelleBodenhoehe	1000	4568
	BezugspunktDach	1000	1001
	measuredHeight	7.070	-7.070
	storeysAboveGround	2	-
	Ableitungsdatum	-	-
	Lagebezeichnung	Siehe Beispieldatensatz_ AdV-CityGML-Profil_LOD1.xml	Keine Änderung vorgenommen

Tabelle 9: Manipulierte Attributwerte des AdV-CityGML-Datensatzes LOD1

Analog zum AdV-CityGML-Datensatz des Detaillierungsgrades LOD1 wurde auch der AdV-CityGML-Datensatz im LOD2 manipuliert. In Tabelle 10 sind die originalen und die manipulierten Attributwerte für den LOD2 und deren Änderungen aufgeführt. Der CityGML-Testdatensatz im LOD2 heißt im Original „Beispieldatensatz_ AdV-CityGML-Profil_LOD2.xml“. Der manipulierte CityGML-Datensatz trägt den Namen „AdV-CityGML-Profil_LOD2_manipulated.xml“.

gml:id=DENW_779fa37b-81a0-4938-999d-b7a4dfbe101d			
Building	Attributname	Attributwert original	Attributwert manipuliert
	function	11_1301	-
	gml_name	Brauhaus	Brauhaus
	Gemeindeschluessel	05512000	253456789
	Ableitungsdatum	2013-11-05	2013-11-05
	Lagebezeichnung	Siehe Beispieldatensatz_ AdV-CityGML-Profil_LOD2.xml	Keine Änderung vorgenommen
gml:id=GUID_1366028267494_173043			
Building Part	Attributname	Attributwert original	Attributwert manipuliert
	DatenquelleDachhoehe	1000	-
	DatenquelleLage	1000	5678
	DatenquelleBodenhoehe	1400	7891
	roofType	3100	-
	measuredHeight	7.236	-7.236
	storeysAboveGround	2	-
gml:id=GUID_1366028267494_173044			
Building Part	Attributname	Attributwert original	Attributwert manipuliert
	DatenquelleDachhoehe	1000	1011
	DatenquelleLage	1000	1566

	DatenquelleBodenhoehe	1400	5656
	roofType	3100	3100
	measuredHeight	7.808	7.808
	storeysAboveGround	2	-

Tabelle 10: Manipulierte Attributwerte des AdV-CityGML-Datensatzes LOD2

Für den manipulierten Beispieldatensatz im Detaillierungsgrad LOD1 ist das Ergebnis wie erwartet. Als Ergebnis des Prüfprotokolls werden die in Tabelle 11 vereinfacht die aufgelisteten Attribute samt Attributwerten im Excel-Prüfprotokoll geführt.

gml:id=DENW_110e8edf-dda2-4130-a564-87b2a3cb3f35		
Building	Attributname	VALUE
	citygml_function	VALUE_IS_MISSING
	gml_name	VALUE_IS_MISSING
	Gemeindeschluessel	VALUE_IS_INCORRECT
	DatenquelleDachhoehe	VALUE_IS_INCORRECT
	DatenquelleLage	VALUE_IS_INCORRECT
	DatenquelleBodenhoehe	VALUE_IS_INCORRECT
	BezugspunktDach	VALUE_IS_INCORRECT
	citygml_measured_height	VALUE_IS_INCORRECT
	citygml_storeysAboveGround	VALUE_IS_MISSING
	citygml_creationDate	VALUE_IS_MISSING
	citygml_address	Siehe Beispieldatensatz_ AdV-CityGML-Profil_LOD1.xml

Tabelle 11: Prüfprotokoll des manipulierten AdV-CityGML-Datensatzes LOD1

Analog zum manipulierten Beispieldatensatz im LOD1 verhält es sich mit dem manipulierten CityGML-Datensatz im Detaillierungsgrad LOD2. Als Ergebnis des Prüfprotokolls werden die in Tabelle 12 aufgelisteten Attribute im Excel-Prüfprotokoll aufgeführt. Zur Veranschaulichung wurden die Ergebnisse komprimiert dargestellt.

gml:id=DENW_779fa37b-81a0-4938-999d-b7a4dfbe101d		
Building	Attributname	VALUE
	citygml_function	VALUE_IS_MISSING
	gml_name	Brauhaus
	Gemeindeschluessel	VALUE_IS_INCORRECT
	citygml_creationDate	2013-11-05
	citygml_adress	Siehe Beispieldatensatz_ AdV-CityGML-Profil_LOD2.xml
gml:id=GUID_1366028267494_173043		
BuildingPart	Attributname	VALUE
	citygml_roof_type	VALUE_IS_MISSING
	citygml_measured_height	VALUE_IS_INCORRECT
	citygml_storeys_above_ground	VALUE_IS_MISSING
	DatenquelleBodenhoehe	VALUE_IS_INCORRECT
	DatenquelleDachhoehe	VALUE_IS_MISSING
	DatenquelleLage	VALUE_IS_INCORRECT
gml:id=GUID_1366028267494_173044		
BuildingPart	Attributname	VALUE
	citygml_roof_type	3100
	citygml_measured_height	7.808
	citygml_storeys_above_ground	VALUE_IS_MISSING
	DatenquelleBodenhoehe	VALUE_IS_INCORRECT
	DatenquelleDachhoehe	VALUE_IS_INCORRECT
	DatenquelleLage	VALUE_IS_INCORRECT

Tabelle 12: Prüfprotokoll des manipulierten AdV-CityGML-Datensatzes LOD2

6.2.2 Testfall 2

Bei den CityGML-Daten des LDBV Bayern wurde nur der Beispieldatensatz für die Detaillierungsstufe LOD1 geprüft. Dieser musste für die Prüfung zuerst aufbereitet werden, da die generischen Attribute nicht nach den Vorgaben der AdV geschrieben waren. Diese beinhalten Leerzeichen bzw. wurden anders geschrieben. Dies war nötig, da der CityGML-Datensatz

sonst nicht hätte geprüft werden können. Beispielsweise war das generische Attribut Datenquelle Dachhöhe als „Datenquelle Dachhoehe“ angegeben, korrekt ist allerdings „DatenquelleDachhoehe“ ohne Leerzeichen. Das Attribut „BezugspunktDach“ wurde in den CityGML-Daten des LDBV mit „Bezugspunkt Dachhoehe“ geschrieben, wobei korrekterweise das Attribut „BezugspunktDach“ heißen müsste. Der ursprüngliche Zustand sieht wie folgt in Abbildung 45 aus:

```
<gml:description>Simple building in LOD1 as blocks model without bal
      cony</gml:description>
  <gml:name>Gebaeudemodell LoD1 der BVV</gml:name>
  <gml:boundedBy>
    <gml:Envelope srsName="EPSG:31468">
      <gml:lowerCorner srsDimension="3">4200000.00 52000000.00
            0.0</gml:lowerCorner>
      <gml:upperCorner srsDimension="3">4700000.00 58000000.00
            3000.0</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <core:cityObjectMember>
    <bldg:Building gml:id="ID_5772158">
      <gml:description>Stand LK: 03.12.2009</gml:description>
      <gml:name>Nähe Doblbachstraße</gml:name>
      <core:externalReference>
        <core:informationSystem>ALKIS</core:informationSystem>
        <core:externalObject>
          <core:uri />
        </core:externalObject>
      </core:externalReference>
      <gen:stringAttribute name="Datenquelle Dachhoehe">
        <gen:value>1000</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Datenquelle Lage">
        <gen:value>1000</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Datenquelle Bodenhoehe">
        <gen:value>2000</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Bezugspunkt Dachhoehe">
        <gen:value>2200</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Hoehe Grund">
        <gen:value>538.673</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Hoehe Dach">
        <gen:value>542.598</gen:value>
      </gen:stringAttribute>
      <gen:stringAttribute name="Gemeindeschluessel">
        <gen:value>175114</gen:value>
      </gen:stringAttribute>
```

```
<bldg:function>1002</bldg:function>
<bldg:roofType>1000</bldg:roofType>
<bldg:measuredHeight uom="#m">3.925</bldg:measuredHeight>
<bldg:storeysAboveGround />
```

Abbildung 45: Auszug des CityGML-Datensatzes des LDBV Bayern („LoD1_CityGML.xml“)

Um die CityGML-Daten des LOD1 prüfen zu können, war eine händische Umbenennung notwendig, damit diese Daten den Anforderungen zur Prüfung entsprechen.

Für die Korrektur des CityGML-Datensatzes wurde der Texteditor Notepad++ verwendet. Folgende generischen Attribute waren betroffen: Datenquelle Dachhöhe, Datenquelle Bodenhöhe, Datenquelle Lage und Bezugspunkt Dach. Anschließend konnten die CityGML-Daten geprüft werden. In Tabelle 13 sind die manipulierten Attributwerte des CityGML-Datensatzes des LDBV Bayern im LOD1 aufgeführt.

gml:id=”ID_5772158”		
Building	Attributname	Attributwert
	function	1002
	gml_name	Nähe Doblbachstraße
	Gemeindeschluessel	175114
	DatenquelleDachhoehe	1000
	DatenquelleLage	1000
	DatenquelleBodenhoehe	2000
	BezugspunktDach	2200
	measuredHeight	3.925
	storeysAboveGround	-
	Ableitungdatum	-
	Lagebezeichnung	Siehe LoD1_CityGML_manipulated.xml

Tabelle 13: Prüfprotokoll des CityGML-Datensatzes des LDBV Bayern im LOD1

Für den CityGML-Datensatz im Detaillierungsgrad LOD1 (LoD1_CityGML_manipulated.xml) wurde die Prüfung vorgenommen. Das Ergebnis wurde zur Veranschaulichung komprimiert in Tabelle 14 dargestellt.

gml:id="DENW_110e8edf-dda2-4130-a564-87b2a3cb3f35“		
Building	Attributname	Attributwert
	citygml_function	VALUE_IS_MISSING (korrekt ist “VALUE_IS_INCORRECT)
	gml_name	Nähe Doblbachstraße
	Gemeindeschluessel	VALUE_IS_INCORRECT (eigentlich korrekter Wert vorhanden)
	DatenquelleDachhoehe	1000
	DatenquelleLage	1000
	DatenquelleBodenhoehe	2000
	BezugspunktDach	2200
	citygml_measured_height	3.925
	citygml_storeys_above_ground	VALUE_IS_MISSING
	citygml_creationDate	VALUE_IS_MISSING
	citygml_address	Siehe Prüfprotokoll_AdV_LoD1.xlsx

Tabelle 14: Prüfprotokoll des manipulierten CityGML-Datensatzes LOD1 des LDBV Bayern

6.2.3 Testfall 3

Der dritte Testfall ist der CityGML-Datensatz aus Thüringen. Dieser wurde durch die AdV zur Verfügung gestellt und liegt im Detaillierungsgrad LOD1, wie aus Abbildung 46 ersichtlich, vor.

```
<gml:name>LoD1_647_5649_1_TH</gml:name>
  <gml:boundedBy>
    <gml:Envelope srsName="urn:adv:crs:ETRS89_UTM32*DE_DHHN92_NH">
      <gml:lowerCorner srsDimension="3">647000.0 5649000.0
            207.6</gml:lowerCorner>
      <gml:upperCorner srsDimension="3">648000.0 5650000.0 228.0</gml:up
            perCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <core:cityObjectMember>
    <bldg:Building gml:id="DETHL51P0000fMAz">
      <core:creationDate>2015-05-06</core:creationDate>
      <core:externalReference>
```

```
    <core:informationSystem>http://repository.gdi-de.org/sche
        mas/adv/citygml/fdv/art.htm#_9100</core:informationSystem>
    <core:externalObject>
      <core:name>DETHL51P0000fMAz</core:name>
    </core:externalObject>
  </core:externalReference>
  <gen:stringAttribute name="BezugspunktDach">
    <gen:value>3000</gen:value>
  </gen:stringAttribute>
  <gen:stringAttribute name="DatenquelleLage">
    <gen:value>1000</gen:value>
  </gen:stringAttribute>
  <gen:stringAttribute name="DatenquelleBodenhoehe">
    <gen:value>1500</gen:value>
  </gen:stringAttribute>
  <gen:stringAttribute name="Gemeindeschluessel">
    <gen:value>16051000</gen:value>
  </gen:stringAttribute>
  <gen:stringAttribute name="DatenquelleDachhoehe">
    <gen:value>1000</gen:value>
  </gen:stringAttribute>
  <bldg:function>31001_1000</bldg:function>
  <bldg:measuredHeight uom="urn:adv:uom:m">5.7</bldg:measuredHeight>
```

Abbildung 46: Auszug des CityGML-Datensatzes der AdV aus Thüringen („LOD1_647_5649_1_TH.gml“)

In Tabelle 15 liegen die Attributwerte des CityGML-Datensatzes aus Thüringen im LOD1 vor.

gml:id=“DETHL51P0000fMAz”		
Building	Attributname	Attributwert
	function	31001_1000
	gml_name	-
	Gemeindeschluessel	16051000
	DatenquelleDachhoehe	1000
	DatenquelleLage	1000
	DatenquelleBodenhoehe	1500
	BezugspunktDach	3000
	measuredHeight	5.7
	storeysAboveGround	-
	Ableitungsdatum	2015-05-06

	Lagebezeichnung	Siehe LOD1_647_5649_1_TH_manipulated.gml

Tabelle 15: Prüfprotokoll des CityGML-Datensatzes aus Thüringen im LOD1

Für den Beispieldatensatz im Detaillierungsgrad LOD1 (LOD1_647_5649_1_TH_manipulated.gml) wurde die Prüfung vorgenommen. Das Ergebnis liegt zur Veranschaulichung komprimiert in der Tabelle 16 vor.

gml:id="DETHL51P0000fMAz“		
Building	Attributname	VALUE
	citygml_function	VALUE_IS_MISSING (eigentlich korrekter Wert vorhanden)
	gml_name	VALUE_IS_MISSING
	Gemeindeschluessel	VALUE_IS_INCORRECT (eigentlich korrekter Wert vorhanden)
	DatenquelleDachhoehe	1000
	DatenquelleLage	1000
	DatenquelleBodenhoehe	1500
	BezugspunktDach	3000
	measuredHeight	5.7
	storeysAboveGround	VALUE_IS_MISSING
	citygml_creationDate	2015-05-06
	citygml_address	Siehe LOD1_647_5649_1_TH_manipulated.gml

Tabelle 16: Prüfprotokoll des manipulierten CityGML-Datensatzes LOD1 aus Thüringen

6.2.3 Bewertung

Die CityGML-Daten für den Testfall 1 wurden verwendet, um die Korrektheit der FME-Workspaces mit den Transformern zu prüfen. Dieser Workspace bildet das Herzstück der Prüfsoftware. Bei den Beispiel-CityGML-Daten der AdV konnten keine Fehler festgestellt werden. Diese sind seit Beginn an mit dem AdV-CityGML-Profil valide. Das Ergebnis der

Prüfung der manipulierten AdV-CityGML-Datensätze im Detaillierungsgrad LOD1 und LOD2 zeigt, dass die Prüfsoftware im Fall der manipulierten AdV-CityGML-Daten bei allen Attributen zu einem korrekten Ergebnis führt.

Hinsichtlich der CityGML-Daten des LDBV Bayern mussten zuvor die generischen Attribute hinsichtlich ihrer Schreibweise korrigiert werden, sonst wären diese Daten nicht prüfbar gewesen. Insgesamt halten die bayerischen CityGML-Daten nicht den inhaltlichen Vorgaben des AdV-CityGML-Profils Stand. Die Gebäudefunktion entspricht inhaltlich nicht den Vorgaben der AdV und somit nicht einem Wert der zur Verfügung gestellten Codeliste. Ein Attributwert für das Attribut *storeysAboveGround*, sowie das Attribut *Ableitungsdatum* samt Attributwert fehlt.

Bei dem CityGML-Datensatz aus Thüringen fehlen die Attribute *gml_name* und *storeysAboveGround* samt Attributwert.

Bei den beiden CityGML-Datensätzen aus Bayern und aus Thüringen fiel auf, dass bei den Attributen *function* und *Gemeindeschluessel* die Prüfsoftware nicht korrekt arbeitet. So wurde ein fehlerhafter Attributwert bei den CityGML-Daten aus Bayern für das Attribut *function* als fehlender Attributwert interpretiert. In den CityGML-Daten aus Thüringen wurde ein eigentlich korrekter Attributwert von der Prüfsoftware als fehlender Attributwert im Prüfprotokoll ausgegeben. In beiden Datensätzen ist aufgefallen, dass die Attributwerte für den Gemeindeschlüssel nicht korrekt erkannt wurden und im Prüfprotokoll jeweils als „VALUE_IS_INCORRECT“ angezeigt werden. Die beiden Fehler konnten im Rahmen der Arbeit nicht mehr behoben werden.

Bei den CityGML-Datensätzen aus Bayern konnte lediglich der LOD1-Datensatz geprüft werden, da der CityGML LOD2-Datensatz als *Solid* modelliert ist und dies die Prüfungssoftware nicht vorsieht. In einem *Solid* sind keine Gebäudeteile modelliert. Für die Überprüfung auf Korrektheit der CityGML-Datensätze für die Detaillierungsstufe LOD2 wird folglich die Modellierung als MultiParts/BuildingsParts vorausgesetzt. Eine Modellierung im LOD2 als Solid im AdV-Profil ist zwar vorgesehen, wird jedoch nicht von dieser Software unterstützt, da eine Überprüfung der Attributwerte auf den Gebäudeteilen laut Prüfnummer 2270 des AdV-Prüfplans vorgenommen wird. Bei dem Detaillierungsgrad LOD2 sind die zu prüfen-

den Attribute auf den BuildingParts vorhanden und diese werden auf die Attribute hin überprüft. Diese Modellierungsart entsprach den Beispieldateien der AdV und wurde so übernommen. Allerdings entsprachen die CityGML-Daten des LDBV Bayern im LOD2 nicht dieser Modellierungsart und somit konnten diese nicht überprüft werden. Jedoch sieht das AdV-Profil unter Umständen auch vor, dass LOD2-CityGML-Daten auch als *Solid* modelliert sein können und die Attribute dann auf dem Building liegen müssen. Für die Prüfung mit einer Software, die analog zur entwickelten Software aufgebaut ist, wird vorausgesetzt, dass man weiß, wie die CityGML-Datensätze für das LOD2 modelliert sind. Hieraus würde sich folglich eine eigene Vorlagendatei, welche dann die Prüfung durchführt, ergeben. Für diese Arbeit war dies aber aus Zeitgründen nicht vorgesehen. Für die Überprüfung der AdV-CityGML-Datensätze zum Test der Prüfsoftware wurden neben den bereits vorhandenen Attributen, auch die Attributwerte der generischen Attribute entfernt.

CityGML-Syntax der Attribute:

```
<bldg:function>1121</bldg:function>
```

CityGML-Syntax der generischen Attribute:

```
<gen:stringAttribute name="DatenquelleBodenhoehe">
        <gen:value>1000</gen:value>
```

Aufgrund der Modellierungsmethode innerhalb der CityGML-Datensätze, ist FME nicht in der Lage, zu erkennen, ob das Attribut vorhanden ist. Deshalb können generische Attribute nicht auf fehlende Attributwerte geprüft werden.

7 Fazit

In diesem Kapitel werden die erreichten Ergebnisse zusammengefasst und eine Bewertung hinsichtlich der erarbeiteten Prüfsoftware geliefert. Zum Schluss folgt ein Ausblick auf mögliche Erweiterungen und Verbesserungen der realisierten Anwendung.

7.1 Bewertung der Arbeit

Im Rahmen dieser Arbeit wurden die Vorgaben der AdV hinsichtlich CityGML-Daten, die Modellierungswerkzeuge mit den Standards der OMG, die interne Sprache des Spatial-ETL-Werkzeuges FME und die Konvertierungsmöglichkeiten von der Modellebene in die Datenebene mit Hilfe der Programmiersprache Python behandelt.
Neben einer Einführung in die theoretischen Grundlagen dieser Arbeit mit den Modellierungsstandards der OMG und CityGML für 3D-Stadtmodelle wurde explizit auf das AdV-CityGML-Profil als Anwendungsbereich eingegangen. Für diesen Anwendungsbereich wurde die Methode der generischen Attribute gewählt.
Im mittleren Teil der Arbeit wurden die Softwarewerkzeuge für die Datenprüfung vorgestellt. Zuerst wurde eine Einführung in das Softwaremodellierungswerkzeug Enterprise Architect der Firma Sparx Systems gegeben. Anschließend wurden die Programmiersprache Python und die relevanten Module für die realisierte Prüfsoftware aufgezeigt. Zu guter Letzt wurde auf das zentrale Werkzeug FME von Safe Software für die Überprüfung der Attributwerte näher eingegangen.
Im darauffolgenden Kapitel wurde die Realisierung des AdV-CityGML-Profils auf Modellebene näher erklärt. Eine Realisierung war mit den Modellierungsstandards der OMG möglich. Mit Hilfe der UML und OCL konnte eine erfolgreiche Verarbeitung vieler Vorgaben der AdV an CityGML-Daten ermöglicht werden. Vorgaben des Prüfplans der AdV, welche nicht in OCL modelliert werden konnten, wurden im weiteren Teil der Arbeit in FME mit Transformern umgesetzt und mussten somit nicht von der Modellebene in die Datenebene überführt werden. Dadurch kommt man dem Ziel, möglichst viele Prüfregeln des Prüfplans der AdV umzusetzen und in der Prüfsoftware zu integrieren, näher.

Anschließend wurde die Prüfsoftware und deren Funktionsweise vorgestellt. Dabei wurde auf die Programmoberfläche, den Aufbau der FME-Workspaces, das Vorlagenkonzept und den Konvertierungsprozess eingegangen.
Schließlich wurde die erstellte Prüfsoftware hinsichtlich der Korrektheit der Überprüfung von AdV-CityGML-Datensätzen evaluiert.
Der Schwerpunkt in der vorliegenden Arbeit lag in der Umsetzung der OCL-Konstrukte und der möglichen Überführung in die interne Sprache der FME.
Ein Teilziel dieser Arbeit war es, diese Werte mit Hilfe der Programmiersprache Python zu zerlegen und in einem Wörterbuch (englisch dictionary) abzuspeichern. Allerdings sind die Werte der Attribute in einer eigens für die OCL vorgegebenen Schreibweise aufgeführt. Diese Schreibweise entspricht weder der Struktur der CityGML-AdV-Modells noch der City-GML-Dateien.
Als Lösung hierfür stand ein weiterer Schritt zum Auslesen der Namen der Attribute, sowie der zugehörigen Werte im Raum. Das Resultat wird in einen FME-Workspace geschrieben und kann ausgeführt werden.
Nicht alle Anforderungen an die CityGML-Daten nach dem Anforderungsprofil der AdV können in OCL umgesetzt werden. Allerdings wurde von denen, die realisiert wurden auch nur ein Teil tatsächlich realisiert. Es konnten die Werte aus den Codelisten in OCL verarbeitet werden, jedoch konnten alle anderen Werte lediglich auf ihre Existenz hin oder einem Attributwert größer Null überprüft werden. Eine vollständige Umsetzung von OCL (Attributwerte und Operatoren) würde den Rahmen dieser Arbeit übersteigen.
Vor dem Hintergrund der unterschiedlichen Anwendungsbereiche von CityGML und der Möglichkeit die zu Verfügung stehenden Datensätze vorab auf Konformität zum Anwendungsbereich zu überprüfen, wurde im Rahmen dieser Arbeit eine Prüfsoftware entwickelt.

7.2 Ausblick

Die im Rahmen dieser Arbeit erstellte Prüfsoftware stellt einen Prototyp für die Überprüfung von CityGML-Datensätzen im LOD1 und LOD2 nach Vorgaben der AdV dar. Aus dieser

Software könnte durch Modifikation eine fertige Software erstellt werden. Mit dieser Software wäre es dann noch komfortabler, CityGML-Datensätze auf Konformität der AdV-Spezifikation zu überprüfen.

Dabei könnte man einen weitaus größeren Teil der Erstellung des FME-Workspaces dynamischer machen um die benötigten Vorlagen zu reduzieren. Möglich wäre es noch, anhand der zu prüfenden Datensätze eine automatische Erkennung von LOD1 und LOD2 vorzunehmen, damit eine Auswahl der „richtigen“ Vorlagen entfallen würde. Andererseits könnte auch eine Prüfung der Detaillierungsstufe LOD2, je nach Modellierungsmethode als *Solid* oder *MultiSurface*, mit Hilfe der Software möglich sein. Ob BuildingParts vorhanden sind und die darauf sich befindenden Attribute geprüft werden, könnte somit automatisch von statten gehen. Damit ließe sich eine weitere Prüfnummer (2270) des AdV-Prüfplans umsetzen.

Als weiteren wichtigen Punkt sei die Umsetzung der Operatoren in UML in Form von OCL-Konstrukten erwähnt, welche auch vollständig (Attributwerte und Operatoren) überführt werden könnten. Einen Beitrag hierzu kann die Promotionsarbeitt von Maximilian Sindram von der Technischen Universität München liefern, welche dieses Thema behandelt.

In dieser Arbeit lag der Schwerpunkt in der Umsetzung der Attributnamen und Attributwerte mit Codelisten von OCL nach FME. Generell ist die Möglichkeit jedoch gegeben, neben den Attributnamen und Attributwerten auch die Operatoren von OCL nach FME zu überführen.

Ein Testen der Software mit anderen Testfällen aus anderen Bundesländern, deren CityGML-Datensätze im LOD1 bereits vorliegen oder deren CityGML-Datensätze im LOD2 der Modellierungsmethode mit *MultiParts* vorgenommen wurde.

Mit FME 2016 wurde ein neuer Transformer vorgestellt. Den Transformer AttributeManager. Dieser Transformer könnte den für diese Arbeit erstellten FME-Workspace vereinfachen.

A Anhang

OCL-Konstrukte der Attribute

Attributname	OCL-Konstrukt
Function	inv: name = 'citygml_function' and (value = 31001_1000 or value = 31001_1010 or value = 31001_1020 or value = 31001_1021 or value = 31001_1022 or value = 31001_1023 or value = 31001_1024 or value = 31001_1025 or value = 31001_1100 or value = 31001_1110 or value = 31001_1120 or value = 31001_1121 or value = 31001_1122 or value = 31001_1123 or value = 31001_1130 or value = 31001_1131 or value = 31001_1210 or value = 31001_1220 or value = 31001_1221 or value = 31001_1222 or value = 31001_1223 or value = 31001_1310 or value = 31001_1311 or value = 31001_1312 or value = 31001_1313 or value = 31001_2000 or value = 31001_2010 or value = 31001_2020 or value = 31001_2030 or value = 31001_2040 or value = 31001_2050 or value = 31001_2051 or value = 31001_2052 or value = 31001_2053 or value = 31001_2054 or value = 31001_2055 or value = 31001_2056 or value = 31001_2060 or value = 31001_2070 or value = 31001_2071 or value = 31001_2072 or value = 31001_2073 or value = 31001_2074 or value = 31001_2080 or value = 31001_2081 or value = 31001_2082 or value = 31001_2083 or value = 31001_2090 or value = 31001_2091 or value = 31001_2092 or value = 31001_2093 or value = 31001_2094 or value = 31001_2100 or value = 31001_2110 or value = 31001_2111 or value = 31001_2112 or value = 31001_2113 or value = 31001_2114 or value = 31001_2120 or value = 31001_2121 or value = 31001_2130 or value = 31001_2131 or value = 31001_2140 or value = 31001_2141 or value = 31001_2142 or value = 31001_2143 or value = 31001_2150 or value = 31001_2160 or value = 31001_2170 or value = 31001_2171 or value = 31001_2172 or value = 31001_2180 or value = 31001_2200 or value = 31001_2210 or value = 31001_2211 or value = 31001_2212 or value = 31001_2213 or value = 31001_2220 or value = 31001_2310 or value = 31001_2320 or value = 31001_2400 or value = 31001_2410 or value = 31001_2411 or value = 31001_2412 or value = 31001_2420 or value = 31001_2421 or value = 31001_2422 or value = 31001_2423 or value = 31001_2424 or value = 31001_2430 or value = 31001_2431 or value = 31001_2440 or value = 31001_2441 or value = 31001_2442 or value = 31001_2443 or value = 31001_2444 or value = 31001_2450 or value = 31001_2451 or value =

	31001_2460 or value = 31001_2461 or value = 31001_2462 or value = 31001_2463 or value = 31001_2464 or value = 31001_2465 or value = 31001_2500 or value = 31001_2501 or value = 31001_2510 or value = 31001_2511 or value = 31001_2512 or value = 31001_2513 or value = 31001_2520 or value = 31001_2521 or value = 31001_2522 or value = 31001_2523 or value = 31001_2527 or value = 31001_2528 or value = 31001_2529 or value = 31001_2540 or value = 31001_2560 or value = 31001_2570 or value = 31001_2571 or value = 31001_2580 or value = 31001_2590 or value = 31001_2591 or value = 31001_2600 or value = 31001_2610 or value = 31001_2611 or value = 31001_2612 or value = 31001_2620 or value = 31001_2621 or value = 31001_2622 or value = 31001_2623 or value = 31001_2700 or value = 31001_2720 or value = 31001_2721 or value = 31001_2723 or value = 31001_2724 or value = 31001_2726 or value = 31001_2727 or value = 31001_2728 or value = 31001_2729 or value = 31001_2732 or value = 31001_2735 or value = 31001_2740 or value = 31001_2741 or value = 31001_2742 or value = 31001_3000 or value = 31001_3010 or value = 31001_3011 or value = 31001_3012 or value = 31001_3013 or value = 31001_3014 or value = 31001_3015 or value = 31001_3016 or value = 31001_3017 or value = 31001_3018 or value = 31001_3019 or value = 31001_3020 or value = 31001_3021 or value = 31001_3022 or value = 31001_3023 or value = 31001_3024 or value = 31001_3030 or value = 31001_3031 or value = 31001_3032 or value = 31001_3033 or value = 31001_3034 or value = 31001_3035 or value = 31001_3036 or value = 31001_3037 or value = 31001_3038 or value = 31001_3040 or value = 31001_3041 or value = 31001_3042 or value = 31001_3043 or value = 31001_3044 or value = 31001_3045 or value = 31001_3046 or value = 31001_3047 or value = 31001_3048 or value = 31001_3050 or value = 31001_3051 or value = 31001_3052 or value = 31001_3053 or value = 31001_3060 or value = 31001_3061 or value = 31001_3062 or value = 31001_3063 or value = 31001_3064 or value = 31001_3065 or value = 31001_3066 or value = 31001_3070 or value = 31001_3071 or value = 31001_3072 or value = 31001_3073 or value = 31001_3074 or value = 31001_3075 or value = 31001_3080 or value = 31001_3081 or value = 31001_3082 or value = 31001_3090 or value = 31001_3091 or value = 31001_3092 or value = 31001_3094 or value = 31001_3095 or value = 31001_3097 or value = 31001_3098 or value = 31001_3100 or value = 31001_3200 or value = 31001_3210 or value = 31001_3211

	or value = 31001_3212 or value = 31001_3220 or value = 31001_3221 or value = 31001_3222 or value = 31001_3230 or value = 31001_3240 or value = 31001_3241 or value = 31001_3242 or value = 31001_3260 or value = 31001_3261 or value = 31001_3262 or value = 31001_3263 or value = 31001_3264 or value = 31001_3270 or value = 31001_3271 or value = 31001_3272 or value = 31001_3273 or value = 31001_3280 or value = 31001_3281 or value = 31001_3290 or value = 31001_9998 or value = 51001_1001 or value = 51001_1002 or value = 51001_1003 or value = 51001_1004 or value = 51001_1005 or value = 51001_1006 or value = 51001_1007 or value = 51001_1008 or value = 51001_1009 or value = 51001_1010 or value = 51001_1011 or value = 51001_1012 or value = 51002_1215 or value = 51002_1220 or value = 51002_1230 or value = 51002_1250 or value = 51002_1260 or value = 51002_1280 or value = 51002_1290 or value = 51002_1330 or value = 51002_1331 or value = 51002_1332 or value = 51002_1333 or value = 51002_1350 or value = 51002_1400 or value = 51003_1201 or value = 51003_1205 or value = 51003_1206 or value = 51006_1430 or value = 51006_1431 or value = 51006_1432 or value = 51006_1440 or value = 51006_1470 or value = 51006_1490 or value = 51007_1210 or value = 51007_1400 or value = 51007_1500 or value = 51007_1510 or value = 51007_1520 or value = 51009_1610 or value = 51009_1611 or value = 51009_1750 or value = 11_1001 or value = 11_1002 or value = 11_1003 or value = 11_1004 or value = 11_1005 or value = 11_1006 or value = 11_1036 or value = 11_1101 or value = 11_1111 or value = 11_1112 or value = 11_1113 or value = 11_1114 or value = 11_1115 or value = 11_1116 or value = 11_1118 or value = 11_1121 or value = 11_1122 or value = 11_1123 or value = 11_1124 or value = 11_1128 or value = 11_1131 or value = 11_1132 or value = 11_1133 or value = 11_1134 or value = 11_1135 or value = 11_1136 or value = 11_1137 or value = 11_1138 or value = 11_1141 or value = 11_1142 or value = 11_1143 or value = 11_1144 or value = 11_1145 or value = 11_1148 or value = 11_1151 or value = 11_1152 or value = 11_1158 or value = 11_1161 or value = 11_1162 or value = 11_1163 or value = 11_1164 or value = 11_1165 or value = 11_1168 or value = 11_1171 or value = 11_1172 or value = 11_1173 or value = 11_1174 or value = 11_1175 or value = 11_1178 or value = 11_1181 or value = 11_1182 or value = 11_1188 or value = 11_1191 or value = 11_1192 or value = 11_1194 or value = 11_1195 or value = 11_1196 or value = 11_1197 or value = 11_1198 or value =

	11_1211 or value = 11_1221 or value = 11_1231 or value = 11_1301 or value = 11_1311 or value = 11_1321 or value = 11_1331 or value = 11_1341 or value = 11_1361 or value = 11_1371 or value = 11_1372 or value = 11_1373 or value = 11_1374 or value = 11_1375 or value = 11_1378 or value = 11_1381 or value = 11_1399 or value = 11_1401 or value = 11_1411 or value = 11_1421 or value = 11_1431 or value = 11_1441 or value = 11_1442 or value = 11_1443 or value = 11_1444 or value = 11_1445 or value = 11_1448 or value = 11_1451 or value = 11_1461 or value = 11_1462 or value = 11_1463 or value = 11_1468 or value = 11_1471 or value = 11_1472 or value = 11_1473 or value = 11_1474 or value = 11_1478 or value = 11_1481 or value = 11_1482 or value = 11_1483 or value = 11_1484 or value = 11_1488 or value = 11_1701 or value = 11_1711 or value = 11_1721 or value = 11_1731 or value = 11_1741 or value = 11_1742 or value = 11_1743 or value = 11_1748 or value = 11_1751 or value = 11_1761 or value = 11_1771 or value = 11_1772 or value = 11_1773 or value = 11_1774 or value = 11_1778 or value = 11_1781 or value = 11_1911 or value = 11_1913 or value = 11_2101 or value = 11_2121 or value = 11_2131 or value = 11_2141 or value = 11_2301 or value = 11_2311 or value = 11_2312 or value = 11_2313 or value = 11_2318 or value = 11_2321 or value = 11_2322 or value = 11_2323 or value = 11_2324 or value = 11_2328 or value = 11_2332 or value = 11_2338 or value = 11_2341 or value = 11_2342 or value = 11_2343 or value = 11_2344 or value = 11_2348 or value = 11_2351 or value = 11_2358 or value = 11_2361 or value = 11_2362 or value = 11_2363 or value = 11_2364 or value = 11_2365 or value = 11_2366 or value = 11_2367 or value = 11_2368 or value = 11_2501 or value = 11_2511 or value = 11_2512 or value = 11_2514 or value = 11_2515 or value = 11_2518 or value = 11_2521 or value = 11_2522 or value = 11_2523 or value = 11_2528 or value = 11_2541 or value = 11_2548 or value = 11_2551 or value = 11_2561 or value = 11_2571 or value = 11_2572 or value = 11_2581 or value = 11_2591 or value = 11_2601 or value = 11_2611 or value = 11_2612 or value = 11_2619 or value = 11_2621 or value = 11_2622 or value = 11_2623 or value = 11_2628 or value = 11_2701 or value = 11_2711 or value = 11_2721 or value = 11_2723 or value = 11_2724 or value = 11_2725 or value = 11_2726 or value = 11_2727 or value = 11_2728 or value = 11_2731 or value = 11_2736 or value = 11_2737 or value = 11_2738 or value = 11_2741 or value = 11_2742 or value = 11_2748 or value = 11_2801 or value = 11_2811 or value =

	11_2812 or value = 11_2818 or value = 11_2821 or value = 11_2822 or value = 11_2828 or value = 11_2831 or value = 11_2841 or value = 11_2842 or value = 11_2848 or value = 11_2851 or value = 11_2861 or value = 11_2862 or value = 11_2863 or value = 11_2868 or value = 11_2871 or value = 11_2872 or value = 11_2873 or value = 11_2874 or value = 11_2878 or value = 11_2881 or value = 11_2882 or value = 11_2883 or value = 11_2888 or value = 11_2891 or value = 11_2894 or value = 11_2921)
Lagebezeichnung	inv: name = ' citygml_address' and value ->notEmpty()
ExternalReference	inv: name = 'externalReference' and value ->notEmpty()
Gebaeudename	inv: name = 'gml_name' and value ->notEmpty()
Ableitungsdatum	inv: name = ' citygml_creationDate' and value ->notEmpty()
Objektidentifikator	inv: name = 'gml_id and value ->notEmpty()
RoofType	inv: name = 'citygml_roof_type' and (value = 1000 or value = 2100 or value = 2200 or value = 3100 or value = 3200 or value = 3300 or value = 3400 or value = 3500 or value = 3600 or value = 3700 or value = 3800 or value = 3900 or value = 4000 or value = 5000 or value = 9999)
MeasuredHeight	inv: name = 'citygml_measured_height' and value > 0
StoreysAboveGround	inv:name = 'citygml_storeys_above_ground' and value > 0
AmtlicherGemeinde-schluessel	inv: name = 'Gemeindeschluessel' and value ->notEmpty()
DatenquelleDachhoehe	inv: name = 'DatenquelleDachhoehe' and (value = 1000 or value = 2000 or value = 3000 or value = 4000 or value = 5000 or value = 6000)
DatenquelleLage	inv: name = 'DatenquelleLage' and (value = 1000 or value = 1100 or value = 1200 or value = 1300 or value = 2000 or value = 3000)
DatenquelleBodenhoehe	inv: name = 'DatenquelleBodenhoehe' and (value = 1000 or value = 1100 or value = 1200 or value = 1300 or value = 1400 or value = 1500 or value = 1600 or value = 1700 or value = 1800 or value = 2000 or value = 3000 or value = 4000)
BezugspunktDachhoehe	inv: name = 'BezugspunktDach' and (value = 1000 or value = 1100 or value = 1200 or value = 1300 or value = 1400 or value = 1500 or value = 1600 or value = 1700 or value = 1800 or value = 2000 or value = 3000 or value = 4000 or value = 9998)

B Anhang

Definierbare Prüfregeln in OCL und FME

Prüfnummer	Testkriterien	definierbar in OCL	definierbar in FME
2100	Keine leeren Attribute	ja	ja
2210	Attribute vollständig?	nein	ja
2220	Korrekte Werte aus der Codeliste generische Attribute	ja	ja
2221	Korrekte Belegung des Attribut Gemeindeschlüssel	nein	ja
2230	Korrekte Werte aus der Codeliste - function	ja	ja
2240	Korrekte Werte aus der Codeliste - rooftype	ja	ja
2250	*minimale Höhe*	ja	ja
2260	*maximale Höhe*	ja	ja
2270	Attribute dem Gebäude oder Gebäudeteil zugeordnet	nein	nein
2280	länderspezifische Attribute vorhanden	nein	nein
2310	BuildingPart gehört nur zu einem Gebäude	nein	nein
2320	BuildingParts Teil eines Gebäudes	nein	nein
2330	keine Unterteilung von BuildingParts	nein	nein
2410	Korrekte OID?	nein	ja
2420	Ist der Dateiname korrekt?	nein	ja
2430	<gml:name> bei Citymodel vorhanden? Sind die Werte zulässig	nein	ja
2510	Pro CityGML-Datei ist ein Envelope zu bilden	nein	nein
2520	CRS nur in Envelope bei CityModel?	nein	nein
2610	Alle Geometrien mit 3 Nachkommastellen? Auch das Envelope	nein	nein
2620	keine Kreisbögen im Datensatz	nein	nein
2630	Keine Referenzen auf Geometrie anderer Objekte	nein	nein

C Anhang

Quellcode des Pythonprogramms in Abschnitten

Abschnitt 1: Notwendige Python-Module für diese Anwendung

```
# -*- coding: utf-8 -*-
# Import aller benötigten Module
import os
import sys
import fmeobjects
import csv
import unittest
import time
import string
import re
import glob
import os.path
from os.path import join
from collections import defaultdict
import Tkinter
from Tkinter import *
import Tkinter as tkinter
from ttk import *
from Tkinter import TclError
import tkMessageBox
from tkFileDialog import *
import tkFileDialog
from tkFileDialog import askopenfilename
import fileinput
import xpath
import lxml
from collections import defaultdict
from lxml import etree
from lxml import objectify
import random
from random import random
from random import randint
import itertools
```

Abschnitt 2: Grafische Oberfläche des Prüftools

```
class App:
        # Initialisierung
        def __init__(self, master):
                self.master = master
                #master = Tkinter.Tk()
                self.start()

        # Titel und Zeit der Applikation
        def start(self):
                self.master.title("AdV-CityGML-Prüftool")

                # Erstellen einer Variable mit Text
                label01 = "XMI-Datei auswählen:"
                Label(self.master, text=label01).grid(row=1, column=1,
sticky=W)

                # Textfeld erstellen
                self.filelocation = Entry(self.master)
                self.filelocation["width"] = 33
                self.filelocation.focus_set()
                self.filelocation.grid(row=1, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.open_file = Button(self.master, text="Browse...",
command=self.browse_file)
                self.open_file.grid(row=1, column=3, sticky=W)


#########################################################################

                # FME Workspace auswählen
                label02 = "FME Vorlage auswählen:"
                Label(self.master, text=label02).grid(row=2, column=1,
sticky=W)

                # Textfeld erstellen
                self.filelocation1 = Entry(self.master)
                self.filelocation1["width"] = 33
                self.filelocation1.focus_set()
                self.filelocation1.grid(row=2, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.open_file1 = Button(self.master, text="Browse...",
command=self.browse_file1)
                self.open_file1.grid(row=2, column=3, sticky=W)


#########################################################################

                # Quelldatensatz auswählen
                label03 = "CityGML-Datensatz auswählen:"
                Label(self.master, text=label03).grid(row=3, column=1,
sticky=W)
```

```
                # Textfeld erstellen
                self.filelocation2 = Entry(self.master)
                self.filelocation2["width"] = 33
                self.filelocation2.focus_set()
                self.filelocation2.grid(row=3, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.open_file2 = Button(self.master, text="Browse...",
command=self.browse_file2)
                self.open_file2.grid(row=3, column=3, sticky=W)


##########################################################################

                # Erstellen einer Variable mit Text
                label04 = "Prüfbericht Pfad anlegen:"
                Label(self.master, text=label04).grid(row=4, column=1,
sticky=W)

                # Textfeld erstellen
                self.directorylocation = Entry(self.master)
                self.directorylocation["width"] = 33
                self.directorylocation.focus_set()
                self.directorylocation.grid(row=4, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.open_directory = Button(self.master,
text="Browse...", command=self.browse_directory)
                self.open_directory.grid(row=4, column=3, sticky=W)


##########################################################################

                # Erstellen einer Variable mit Text
                label05 = "Prüfbericht Name anlegen:"
                Label(self.master, text=label05).grid(row=5, column=1,
sticky=W)

                # Textfeld erstellen
                self.filelocation3 = Entry(self.master)
                self.filelocation3["width"] = 33
                self.filelocation3.focus_set()
                self.filelocation3.grid(row=5, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.save_filename3 = Button(self.master, text="Save...",
command=self.save_file)
                self.save_filename3.grid(row=5, column=3, sticky=W)


##########################################################################

                # Erstellen einer Variable mit Text
                label06 = "CityGML-Prüfung Name anlegen:"
```

```
                Label(self.master, text=label06).grid(row=6, column=1,
sticky=W)

                # Textfeld erstellen
                self.filelocation4 = Entry(self.master)
                self.filelocation4["width"] = 33
                self.filelocation4.focus_set()
                self.filelocation4.grid(row=6, column=2, sticky=W)

                # Button erstellen, um Datei zu öffnen
                self.save_filename4 = Button(self.master, text="Save...",
command=self.save_file1)
                self.save_filename4.grid(row=6, column=3, sticky=W)


#########################################################################

                # Button erstellen, um Programm zu beenden
                self.bu = Button(self.master, text="Beenden", com-
mand=self.beenden)
                self.bu.grid(row=7, column=1, sticky=W)

                # Button erstellen, um Konvertierung auszuführen
                self.submit = Button(self.master, text="Konvertierung
starten", command=self.start_processing)
                self.submit.grid(row=7, column=2, sticky=W)

                # Button erstellen, um Konvertierung auszuführen
                self.submit = Button(self.master, text="FME starten",
command=self.start_processing1)
                self.submit.grid(row=7, column=2, sticky=E)


#########################################################################

        # Funktion eine Datei zu öffnen
        def browse_file(self):

                # Name der Datei in self.filename speichern
                self.filename = tkFileDialog.askopenfilename(title="Datei
öffnen...", filetypes=[("XML Metadata Interchange-file","*.xmi"), ("eX-
tensible Markup Language file","*.xml")])

                # Setzt den Text von Dateipfad in Textfeld
                self.filelocation.insert(0,self.filename)


#########################################################################

        # Funktion eine Datei zu öffnen
        def browse_file1(self):

                # Name der Datei in self.filename speichern
                self.filename1 = tkFileDialog.askopenfilename(title="Da-
tei öffnen...", filetypes=[("FMW Workbench Files","*.fmw")])
```

```
                # Setzt den Text von Dateipfad in Textfeld
                self.filelocation1.insert(0,self.filename1)


#########################################################################

        # Funktion eine Datei zu öffnen
        def browse_file2(self):

                # Name der Datei in self.filename speichern
                self.filename2 = tkFileDialog.askopenfilename(title="Da-
tei öffnen...", filetypes=([("eXtensible Markup Language file","*.xml"),
("eXtensible Markup Language file","*.gml")]))

                # Setzt den Text von Dateipfad in Textfeld
                self.filelocation2.insert(0,self.filename2)


#########################################################################

        # Funktion eine Datei zu öffnen
        def browse_directory(self):

                # Name der Datei in self.filename speichern
                self.directoryname = tkFileDialog.askdirectory(ti-
tle="Pfad öffnen...", initialdir=("C:\\"))

                # Setzt den Text von Dateipfad in Textfeld
                self.directorylocation.insert(0,self.directoryname)


#########################################################################

         # Funktion eine Datei zu öffnen
        def save_file(self):

                # Name der Datei in self.filename speichern
                self.filename3 = tkFileDialog.asksaveasfilename(ti-
tle="Speichern...", filetypes=([("Excel Workbook","*.xlsx")]))

                # Setzt den Text von Dateipfad in Textfeld
                self.filelocation3.insert(0,self.filename3)

#########################################################################

         # Funktion eine Datei zu öffnen
        def save_file1(self):

                # Name der Datei in self.filename speichern
                self.filename4 = tkFileDialog.asksaveasfilename(ti-
tle="Speichern...", filetypes=[("FMW Workbench Files","*.fmw")])

                # Setzt den Text von Dateipfad in Textfeld
                self.filelocation4.insert(0,self.filename4)
```

```
#########################################################################
```

Abschnitt 3: Konvertierungsprozess XMI

```
        # Start der Konvertierung von XMI in ein FME Workbench File
        def start_processing(self):

                try:
                        # open AdVCityGML_LoD2_2.4.1.xmi
                        cons_in = etree.parse(self.filename)

                        # Vorhandes template lesen
                        fme_in = open(self.filename1, 'r')

                        # beschriebenes template rausschreiben
                        fme_out = open(self.filename4, 'a')

                        # valuelist
                        oclvalues = {}

                  # parsen der OCL-Konstrukte unter "ownedRule"
                        for con in cons_in.xpath("//specification"):

                            for cons in con.xpath("./@body"):
```

Abschnitt 4: Aufbau des Dictionaries

```
                                        # read name and values -> dictionary
                                        ocl = cons.split("'")

                                        if len(ocl) > 1:
                                            if 'notEmpty()' in ocl[2]:
                                                    oclvalues[ocl[1]] =
'notEmpty()'

                                            else:
                                                    values =
ocl[2][ocl[2].find("(")+1:ocl[2].rfind(")")].split("or")
                                                    valuelist=[]
                                                    for value in values:
                                                            valuelist.ap-
pend(value[value.find("=")+1:].strip())
                                                    oclvalues[ocl[1]] = valu-
elist
```

Abschnitt 5: Herausfilterung der Attribute mit Codelisten

```
# Pfade werden in Variable gespeichert und Dateinamnen extrahiert. Temp-
late wird Zeile für Zeile ausgelesen und Variablen der FME-Vorlage durch
Dateinamen ersetzt.
                        a = self.filelocation3.get()
                        alist = a.split('/')
```

```
laenge = len(alist)
afilename = alist[laenge-1]
b = self.filelocation2.get()
blist = b.split('/')
laengeb = len(blist)
bfilename = blist[laenge-1]

lines = fme_in.readlines()

for line in lines:

        pline = line.replace('%w%', afilename)

        oline = pline.replace('%u%', bfilename)

        fme_out.writelines(oline)

fme_in.close()

for key, value in oclvalues.items():

        # Remove keys with value ='notEmpty()'
        if value != 'notEmpty()':

                del oclvalues[key]
```

Abschnitt 6: Erzeugen der dynamischen Codezeilen der FME-Transformer

```
# zählt von 1 auf 2 für unterschiedliche Numerierung der Factories und
Filter
                                        for i in range (1):

                                                if i < 1:

                                                        i = i + 1

                                                # 2 für unterschiedliche
Numerierung der Testfilterinputline
                                                for e in range (2, 3, 1):

                                                        zeile1 = 'FAC-
TORY_DEF * TestFactory FACTORY_NAME TestFilter_' + str(key) + '_TestFac-
tory_' + str(i) + ' INPUT FEATURE_TYPE TestFilter_' + str(key) + '_TEST-
FILTERINPUTLINE_' + str(i) + ' TEST @EvaluateExpression(FDIV,STRING_EN-
CODED,<at>Value<openparen>' + str(key) + '<closeparen>,TestFilter_' +
str(key) + ') NOT_IN '

                                                        werte = ''

                                                        for valuex in
value:
```

```
                                                                werte =
werte + valuex + '<comma>'

                                                        werte =
werte[0:len(werte)-7] + ' ENCODED BOOLEAN_OPERATOR OR COMPOSITE_TEST_EXPR
<Unused> OUTPUT PASSED FEATURE_TYPE TestFilter_' + str(key) +
'_<at>Value<openparen>' + str(key) + '<closeparen><space>'

                                                        zeileteil1 =
zeile1 + werte

                                                        zeile2 =
'NOT_IN<space> '

                                                        werte = ''

                                                        for valuex in
value:

                                                                werte =
werte + valuex + '<comma>'

                                                        werte =
werte[0:len(werte)-7] + ' OUTPUT FAILED FEATURE_TYPE TestFilter_' +
str(key) + '_TESTFILTERINPUTLINE_' + str(e)

                                                        zeileteil2 =
zeile2 + werte

                                                        # zusammensetzen
des ersten Teils des dynamischen Teils der FME Workbench
                                                        zeilegesamt =
zeileteil1 + zeileteil2

                                                        if 'ci-
tygml_storeys_above_ground' not in zeilegesamt and 'citygml_mea-
sured_height' not in zeilegesamt:


fme_out.write(zeilegesamt + '\n')

                                                        #Zufallszahl für
AttributeCreator zwischen 9 und 15
                                                        f = rand-
int(9,15+1)

                                                        zeile3 = 'FAC-
TORY_DEF * AttrSetFactory FACTORY_NAME AttributeCreator_' + str(f) + '
ATTRSET_CREATE_DIRECTIVES _PROPAGATE_MISSING_FDIV INPUT  FEATURE_TYPE
TestFilter_' + str(key) + '_<at>Value<openparen>' + str(key) +
'<closeparen><space>NOT_IN<space>'

                                                        werte = ''
```

```
                                                        for valuex in
value:

                                                            werte =
werte + valuex + '<comma>'

                                                        werte =
werte[0:len(werte)-7] + ' ATTR ' + str(key) + ' VALUE_IS_INCORRECT
OUTPUT OUTPUT FEATURE_TYPE AttributeCreator_' + str(f) + '_OUTPUT'

                                                        zeileteil3 =
zeile3 + werte

                                                        # dynamischen
Teil um überflüssige Codezeilen bereinigen
                                                        if 'citygml_sto-
reys_above_ground' not in zeileteil3 and 'citygml_measured_height' not in
zeileteil3:

                                                            # raus-
schreiben des gesamten dynamischen Teils der FME Workbench

fme_out.write(zeileteil3 + '\n')

                        fme_out.close()

                        tkMessageBox.showinfo("Info", "XMI konvertiert &
Vorlage gefüllt")

                except:

                        tkMessageBox.showinfo("Info", "XMI konnte nicht
konvertiert & Vorlage nicht gefüllt werden!")
                        sys.exit(0)


#########################################################################
```

Abschnitt 7: Starten der FME über Button in der Programmoberfläche

```
        # Starten von FME  - nicht mehr realisiert

        def start_processing1(self):

                try:
                        #Schreiben der FME Workbench-Ausführungsdatei

                        runworkspace_out = open('RunWorkspace.py', 'w')

                        runworkspace_out.write("import sys"  '\n')

                        runworkspace_out.write("import fmeobjects"  '\n')
```

```
                        runworkspace_out.write('sys.path.append("C:\\Pro-
gram Files (x86)\\FME\\fmeobjects\\python27")'  '\n')

                        runworkspace_out.write("runner = fmeob-
jects.FMEWorkspaceRunner() "  '\n')

                        runworkspace_out.write("workspace ='" +
self.filelocation4.get() + "'"  '\n')

                        runworkspace_out.write("parameters = {}"  '\n')

                        runworkspace_out.write("parameters['SourceDa-
taset_CITYGML_3'] ='" + self.filelocation2.get() + "'"  '\n')

                        runworkspace_out.write("parame-
ters['DestDataset_XLSXW2_4'] ='" + self.filelocation3.get() + "'"  '\n')

                        runworkspace_out.write("try:"  '\n')

                        runworkspace_out.write("    runner.runWithParame-
ters(workspace, parameters)"  '\n')

                        runworkspace_out.write("except fmeobjects.FMEEx-
ception as ex:"  '\n')

                        runworkspace_out.write("    print ex.message"
'\n')

                        runworkspace_out.write("else:"  '\n')

                        runworkspace_out.write("    print('The Workspace
%s ran successfully'.format(workspace))"  '\n')

                        runworkspace_out.write("runner = None"  '\n')

                        runworkspace_out.close()

                        tkMessageBox.showinfo("Info", "FME wird ges-
tartet")

                        # Starten von FME
                        #os.system(r'fme.exe self.filelocation4 --Source-
Dataset_CITYGML "self.filelocation2" --DestDataset_XLSXW2 "self.fileloca-
tion3"')
                        os.system("RunWorkspace.py 1")

                except:

                        tkMessageBox.showinfo("Info", "FME kann nicht ge-
startet werden")
                        sys.exit(0)


#########################################################################
```

Abschnitt 8: Schließen des Programms

```
        # Beenden des Programms

        def beenden(self):

                self.master.destroy()


#########################################################################

root =root = Tk()
app = App(root)
root.mainloop()
```

Glossar

AAA	Konzeptuelle Anwendungsschema für die Informationssysteme → AFIS, → ALKIS und → ATKIS, die durch die Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder (AdV) neu konzipiert wurden.
ADE	Application Domain Extension
AdV	Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland
AED-SICAD	AED-SICAD AG
AFIS	Amtliches Festpunkt Informationssystem
AGeoBW	Amt für Geoinformationswesen der Bundeswehr
ALKIS	Amtliches Liegenschaftskatasterinformationssystem
ATKIS	Amtliches Topografisches Katasterinformationssytem
CityGML	City Geography Markup Language
CityGML SWG	CityGML Standard Working Group
EBNF	Erweiterte Backus-Naur-Form
FME	Feature Manipulation Engine (Firma Safe Software Inc.)
GDI-DE	Geodateninfrastruktur Deutschland
GeoInfoDok	Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens
GML	Geographic Markup Language
ISO	International Standardisation Organisation.
LOD	Level of Detail
MOF	Meta Object Facility
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OMG	Object Management Group
SIG3D	Special Interest Group 3D
TCL	Tool Command Language
UML	Unified Modeling Language

XMI	XML Metadata Interchange, XML-Austauschformat der OMG
XML	Extensible Markup Language
ZSHH	Zentrale Stelle Hauskoordinaten, Hausumringe und 3D-Gebäudemodelle

Literaturverzeichnis

AdV (2015a): AdV-CityGML-Profile für 3D-Gebäudemodelle. Version 1.3.

AdV (2015b): Datenformatbeschreibung 3D-Gebäudemodell LoD1 Deutschland. Version 1.1.

AdV (2015c): Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens (GeoInfoDok). Hauptdokument - Version 7.0.2 V.

AdV (2015d): Prüfplan für Gebäudemodelle LoD1 / LoD2. Version 1.2.

Banfi, Daniel (2013): Energiebedarfsanalyse urbaner Räume anhand des semantischen Modells und Austauschformats CityGML.

Becker, Oliver (2004): Serielle Transformationen von XML. Mathematisch-Naturwissenschaftliche Fakultät II. Online verfügbar unter urn:nbn:de:kobv:11-10035056.

Behnel, Stefan (2015): lxml.

Casanova, M., Wallet, T., & D'Hondt, M. (Hg.) (2000): Ensuring quality of geographic data with UML and OCL. Proceedings of the 3rd international conference on The unified modeling language: advancing the standard: Springer-Verlag.

Clark, Tony; Warmer, Jos B. (2002): Object Modeling with the OCL. Berlin, London: Springer (Lecture Notes in Computer Science, 2263).

con terra GmbH (2015): FME Desktop. Das deutschsprachige Handbuch für Einsteiger und Anwender. Berlin, Offenbach: Wichmann.

Daly, Liza (2011): High-performance XML parsing in Python with lxml. Stretch the limits of this full-featured XML parsing and serializing suite.

Ernesti, Johannes; Kaiser, Peter (2009): Python 3. Das umfassende Handbuch ; [Einstieg, Praxis, Referenz ; Sprachgrundlagen, Objektorientierung, Modularisierung ; Migration, Debugging, Interoperabilität mit C, GUIs, Netzwerkkommunikation u.v.m. ; Migration von Python 2.x auf 3]. 2., aktualisierte Aufl. Bonn: Galileo (Galileo Computing).

Gozdz et al. (Hg.) (2011): The possibilities of using CityGML for 3D representation of buildings in the cadastre. Proceedings 4th International Workshop on 3D Cadastres, 9-11 November 2014, Dubai, United Arab Emirates: International Federation of Surveyors (FIG).

Gröger, G.; Kolbe, T. H.; Nagel, C.; Häfele, K. H. (2012): OGC city geography markup language (CityGML) encoding standard, Version 2.0, OGC doc no. 12-019. In: *Open Geospatial Consortium*.

Gröger, Gerhard; Plümer, Lutz (2012): CityGML – Interoperable semantic 3D city models. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 71, S. 12–33. DOI: 10.1016/j.isprsjprs.2012.04.004.

Gröger et al. (2005): Das interoperable 3D-Stadtmodell der SIG 3D. In: *Zeitschrift für Vermessungswesen* 130 (6), S. 343–353.

Hussmann, Heinrich; Demuth, Birgit; Finger, Frank (2002): Modular architecture for a toolset supporting OCL. In: *Science of Computer Programming* 44 (1), S. 51–69. DOI: 10.1016/S0167-6423(02)00032-1.

Jacinto, Marta Henriques; Librelotto, Giovani Rubert; Ramalho, JosÃ© Carlos; Henriques, Pedro Rangel (2002): Constraint specification languages: comparing XCSL, Schematron and XML-Schemas.

Kargl, Horst; Steinpichler, Dietmar (2015): Kompendium zu Enterprise Architect von SparxSystems. Trainingsunterlage. überarbeitete Auflage für EA 12.

Kecher, Christoph (2009): UML 2. In: *Das umfassende Handbuch. Bonn*.

Kolbe, Thomas H. (2009): Representing and exchanging 3D city models with CityGML. In: Lee, Jiyeong, Zlatanova, Siyka (Hg.): 3D geo-information sciences: Springer, S. 15–31.

Kolbe, Thomas H.; Kutzner, Tatjana; Gröger, Gerhard; Casper, Egbert (2014): CityGML - Der OGC-Standard CityGML geht in die nächste Runde. In: *gis@work* 1, S. 42–43.

Kutzner, Tatjana; Eisenhut, C. (2010): Vergleichende Untersuchungen zur Modellierung und Modelltransformation in der Region Bodensee im Kontext von INSPIRE. In: *Technische Universität München*.

Kutzner et al. (2012): INSPIRE auf dem Prüfstand der grenzüberschreitenden Praxistauglichkeit in der Testregion Bodensee.

Lawhead, Joel (2013): Learning Geospatial Analysis with Python: Packt Publishing.

Louwsma, Jildou; Zlatanova, Sisi; van Lammeren, Ron; van Oosterom, Peter (2006): Specifying and Implementing Constraints in GIS—with Examples from a Geo-Virtual Reality System. In: *Geoinformatica* 10 (4), S. 531–550. DOI: 10.1007/s10707-006-0345-5.

Löwner, Marc-O; Benner, Joachim; Gröger, Gerhard; Gruber, Ulrich; Häfele, Karl-Heinz; Schlüter, Sandra (2012): CityGML 2.0-ein internationaler Standard für 3D-Stadtmodelle, Teil 1: Datenmodell. In: *Zeitschrift für Geodäsie, Geoinformation und Landmanagement* 6 (2012), S. 340–349.

Löwner, Marc-O; Casper, Egbert; Becker, Thomas; Benner, Joachim; Gröger, Gerhard; Gruber, Ulrich et al. (2013): CityGML 2.0–ein internationaler Standard für 3D-Stadtmodelle, Teil 2: CityGML in der Praxis. In: *Zeitschrift für Geodäsie, Geoinformation und Landmanagement* 2 (2013), S. 131–143.

McNeil, Jeff (2010): Python 2.6 text processing. Beginner's guide : the easiest way to learn how to manipulate text with Python. Birmingham, U.K.: Packt Open Source.

Mertz, David (2003): Text processing in Python. Boston: Addison-Wesley.

Oestereich, B.; Bremer, S. (2012): Analyse und Design mit der UML 2.5: Objektorientierte Softwareentwicklung: Oldenbourg Wissenschaftsverlag. Online verfügbar unter https://books.google.de/books?id=7DAys9ReO1YC.

Oestereich, Bernd; Scheithauer, Axel (2014): Die UML-Kurzreferenz 2.5 für die Praxis. Kurz, bündig, ballastfrei. 6. Aufl. München: Oldenbourg.

OMG (2014a): OMG Object Constraint Language. Version 2.4. OMG.

OMG (2014b): XML Metadata Interchange (XMI) Specification. Version 2.4.2. OMG.

OMG (2015): OMG Unified Modelling Language (OMG UML). Version 2.5. OMG.

Rumpe, Bernhard (2012): Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring: Springer-Verlag.

Safe Software Inc. (2007): FME Fundamentals.

Safe Software Inc. (2008): FME Functions and Factories: Reference for Functions and Factories supported by the Feature Manipulation Engine. FME Documentation, Help File.

Safe Software Inc. (2009a): GIS Interoperability: The Safe Way.

Safe Software Inc. (2009b): Readers and Writers:

Safe Software Inc. (2014): Python-FME Manual.

Safe Software Inc. (2015a): FME Desktop Training Manual.

Safe Software Inc. (2015b): FME Desktop Under the Hood.

Sparks, Geoffrey (2012): Enterprise Architect User Guide.

Stadler, A., & Kolbe, T. H. (Hg.) (2007): Spatio-semantic coherence in the integration of 3D city models. Proceedings of the 5th International Symposium on Spatial Data Quality, Enschede.

Steinpichler, Dietmar; Kargl, Horst (2012): Project development with UML and Enterprise Architect. Training documentation 9.2. rev. ed. Wien: SparxSystems Software (Enterprise Architect).

Störrle, Harald (2005): UML 2 erfolgreich einsetzen. Einführung und Referenz ; [mit farbigem Notationsposter ; aktuelle UML-Tools auf CD]. München, Boston [u.a.]: Addison-Wesley (Programmer's choice).

Theis, Thomas (2011): Einstieg in Python. [ideal für Programmieranfänger geeignet ; Schritt für Schritt eigene Programme entwickeln ; mit vielen Beispielen und Übungsaufgaben ; inkl. Benutzeroberflächen, objektorientierte Programmierung, Datenbank- und Internetanwendungen u.v.m. ; aktuell zu Python 3.2 und Python 2.7]. 3., aktualisierte Aufl. Bonn: Galileo Press (Galileo Computing).

Tidwell, Doug (2008): XSLT. 2nd ed. Sebstopol, CA: O'Reilly Media.

Tilch, Sebastian (2008): Vergleich der Transformationssprache UMLT mit den Möglichkeiten der Software FME zur Semantischen Transformation Topographischer Informationssysteme im Kontext von INSPIRE. Bachelor's Thesis.

Wagner, Detlev; Kolbe, Thomas H.; Coors, Volker (2014): Spezifikation von Prüfplänen und Prüfergebnissen zur Validierung von 3D-Stadtmodellen.

Wagner, Detlev; Wewetzer, Mark; Bogdahn, Jürgen; Alam, Nazmul; Pries, Margitta; Coors, Volker (2013): Geometric-semantical consistency validation of CityGML models.

In: Pouliot, J., Daniel, S., Hubert, F., Zamyadi, A. (Hg.): Progress and New Trends in 3D Geoinformation Sciences: Springer, S. 171–192.

Warmer, Jos B.; Kleppe, Anneke G. (2003): Object Constraint Language, The: Getting Your Models Ready for MDA, Second Edition: Addison-Wesley Professional.

Werder, S. (Hg.) (2009): Formalization of spatial constraints. Proceedings of the 12th AGILE International Conference on Geographic Information Science, Jun: Citeseer.

Internetquellen:

Safe Software Online (2015): Rennie: How does FME handle null attribute values? FME Knowledge Center. Online verfügbar unter https://knowledge.safe.com/articles/21423/how-does-fme-handle-null-attribute-values.html, Aufrufdatum: 03.04.2016.

SIG3D Online (2016): Aufgabenschwerpunkte der SIG3D. Online verfügbar unter http://www.sig3d.org/index.php?catid=2&themaid=1961060, Aufrufdatum: 19.04.2016

Abbildungsverzeichnis

Tabellenverzeichnis