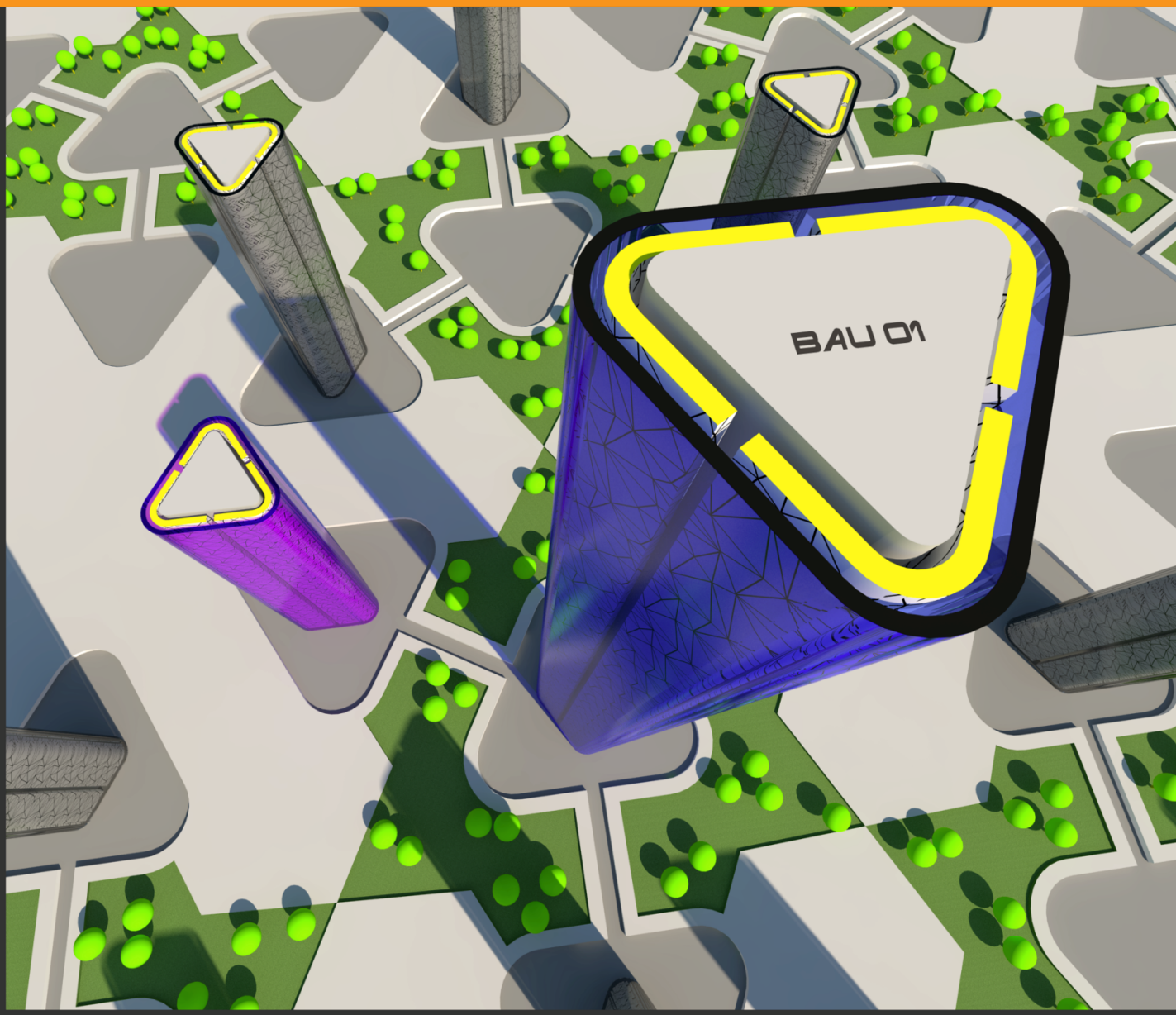
Hochschule
für Technik
Stuttgart



Evaluation of validation approaches for 3D city models & Integration of a new quality feature into CityServer3D

Master Thesis

Student
Athanasios Koukofikis

Supervisors
Prof. Dr.-Ing. Volker Coors
MSc. Inf. Michel Krämer

{ March 2014 }

Hochschule
für Technik
Stuttgart

Evaluation of validation and healing approaches for 3D city models & Integration of a new healing feature into CityServer3D

by

Athanasios Koukofikis

A dissertation presented in partial fulfillment of the requirements for the degree of Master of Science in the Department of Geomatics, Computer Science and Mathematics, Stuttgart University of Applied Sciences

Declaration

The following Master thesis was prepared in my own words without any additional help. All used sources of literature are listed at the end of the thesis.

Stuttgart, 28.02.2014

_____

Athanasios Koukofikis

Approved by:

_____

Prof. Dr.-Ing. Volker Coors

Acknowledgement

First I would like to thank God for gracing and giving me health and strength for completing this course. The project would not have been completed without the help of several people.

I want to thank both my supervisors, Prof. Dr.-Ing. Volker Coors and MSc. Inf. Michel Krämer, for their guidance and supervision. Many thanks to Detlev Wagner, Nazmul Alam and Simon Thum for their help. A great appreciation goes to the professors of this master course and Beate Baur, the Programme Coordinator.

I want to say to my colleagues that we were a great team. Eighteen months passed by so fast. This indicates the great mentality and fun we had during this period. Special thanks to Sergey Voinov.

I dedicate this thesis to Persefoni and Katia.

Abstract

CityGML is an OGC standard used to store and exchange 3D city models. Quality assessment of City Models is a key procedure to ensure delivered products and 3D cadastre databases conform to CityGML requirements. Evaluation of specialized tools designed for CityGML validation is an essential need to study the different approaches been used. CityDoctor and FME are evaluated by a benchmark definition where results are stored and analyzed.

Contents

Table of Figures

Table of Tables

Table of Snippets

Table of Graphs

Table of UML Diagrams

Abbreviations

XML	Extensible Markup Language
OGC	Open Geospatial Consortium
GML	Geography Markup Language
CityGML	City Geography Markup Language
2D	Two Dimensional

3D	Three Dimensional
XSD	XML Schema Definition
DTD	Document Type Definition
JDT	Java Development Tools
LOD	Level of Detail
StAX	Streaming API for XML
SAX	Simple API for XML
JAXB	Java Architecture for XML Binding
AdV	Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland
NaN	Not a Number
UML	Unified Modeling Language

1 Introduction

1.1 Geometry

"ἀγεωμέτρητος μηδεὶς εἰσίτω" (literally: "If you are not aware of geometry do not enter") was placed at the entrance of Plato's Academy, 378 BC.

Geometry is the oldest branch of mathematics (Ostermann & Wanner, 2012). The inscription at the entrance of Plato's Academy shows the importance of geometry during this era (Figure 1.1). Geometry studies the shape, size and position of 2D shapes and 3D objects.

Euclid (325 BC) is the father of geometry. His inheritance is 13 books called "The Elements". Among other great geometers are Pappus from Alexandria, and Blaise Pascal. The great Gaspard Monge was the inventor of Descriptive Geometry. Later his student, Jean-Victor Poncelet, while imprisoned in Russia fighting on the side of Napoleon, started to write the first chapters of his book: "Traité des propriétés projectives des figures".

Figure 1.1: Plato walks alongside Aristotle. (Raphael, 1509)

All these pioneers defined the theoretical basis for a variety of geometry applications. Modeling from single objects to whole cities implements the theoretical geometry background.

City models have allowed us to gradually move one step closer to reality with 3D design. 3D city models have become a necessity for analysis, simulation, navigation and decision making.

CityGML is a standard for storing, exchanging and visualizing 3D city models. CityGML is based on a markup language called eXtensible Markup Language (XML).

1.2 XML

XML stands for eXtensible Markup Language. It is a W3C standard which allows the storage and exchange of arbitrary textual data. XML became a recommendation back in 1998 by the Wide Web Consortium.

Terminology

XML: "Is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax used to define other domain-specific, semantic, structured markup languages."

(Harold, 2001)

Extensible Markup Language (XML) is based on simple, platform-independent rules for representing structured textual information (Vohra, Vohra, & Gowda, 2006). Any XML-based file has to conform to the XML syntax rules. An XML parser is not able to parse a file with syntactical errors.

XML uses tab-based syntax which makes it look similar with HTML. The success for this format comes with the ability for the user to use predefined tags or to create his own tags. That is why it can be described as a language to create tag sets.

The main advantage of XML is interoperability (Fawcett, Quin, Ayers, & Tegtmeyer, 2012). The exchange of data is more efficient when XML is used. XML is mainly used to define information structures. Its hierarchical nature allows implementation in a majority of applications.

Additional rules can be defined that apply restrictions to the structure and content of an arbitrary XML file. This extension mechanism is realized by the use of XML Schemas.

1.3 XML Validation

An XML document has to conform to the well-formed rules and the valid content rules. The well-formed rules are global and apply to all types of XML documents.

Elements in XML have a parent-child relationship. A root element has to contain all the child elements. Tags are always case sensitive and should not overlap with other tags. A tag name should not be empty or contain special characters such as: "<" and "&". Moreover, the name of a start tag should match the name of the corresponding end tag.

The freedom XML gives to a user to create his own tags can lead to an interoperability disaster. In order to control the content and the sequence of the elements inside XML, two technologies are used: DTD and XML Schema.

DTD stands for Data Table Definition. Inside a Data Table Definition a user can define the constraints that should be applied to an XML file. The allowed elements and their attributes may exist in a DTD. It can exist inside an XML file or outside as a standalone file. The file extension of DTD is ".dtd". Snippet 1.1 shows a basic example of a DTD.

```
1   <!DOCTYPE note
2   [
3   <!ELEMENT note (to,from,heading,body)>
4   <!ELEMENT to (#PCDATA)>
5   <!ELEMENT from (#PCDATA)>
6   <!ELEMENT heading (#PCDATA)>
7   <!ELEMENT body (#PCDATA)>
8   ]>
```

Snippet 1.1: A simple DTD file. (w3schools.com)

XML Schema is another technology to help control the contents of an XML file. It is more popular because its syntax is based on XML. The file extension of a schema is ".xsd". In general, XML Schemas offer more functionality. It is usually used as an external resource and not inline an XML file. It can also define a complex schema definition by importing other schemas. The imported schemas can also import other schema definitions in a way to create a complex tree schema definition.

Terminology

XML Schema: "Is the W3C-recommended schema definition language, expressed in XML 1.0 syntax, which is intended to describe the structure and constrain the content of documents written in XML."

(Wyke & Watt, 2002)

A simple example of an XSD Schema can be seen in Snippet 1.2.

```
1     <xs:element name="note">
2
3     <xs:complexType>
4     <xs:sequence>
5     <xs:element name="to" type="xs:string"/>
6     <xs:element name="from" type="xs:string"/>
7     <xs:element name="heading" type="xs:string"/>
8     <xs:element name="body" type="xs:string"/>
9     </xs:sequence>
10    </xs:complexType>
11
12    </xs:element>
```

Snippet 1.2: A simple XSD file. (w3schools.com, 2014)

1.4 CityGML

CityGML stands for City Geography Markup Language and is an official OGC standard. CityGML is a GML Application Domain which extends the GML (Geography Markup Language) definition (Figure 1.2). GML, as a markup language, extends by definition the XML language. An analogy, in Object Oriented Programming, would describe GML as an abstract class and CityGML as the extending concrete class.

Terminology

CityGML: An open data model and XML-based format for the storage and exchange of virtual 3D city models.

(OGC, 2012)

CityGML is used to store, represent and exchange 3D city models. During the last decades, numerous formats introduced to store 3D geometry. A special characteristic of CityGML is the ability to feature semantic information for city objects. If a city object in CityGML could be described as a human being, then the "first name" would be the geometry of that object and the "last name" would be the semantic information, for example "roof surface". CityGML can also store simultaneously multiple levels of detail for a city object.

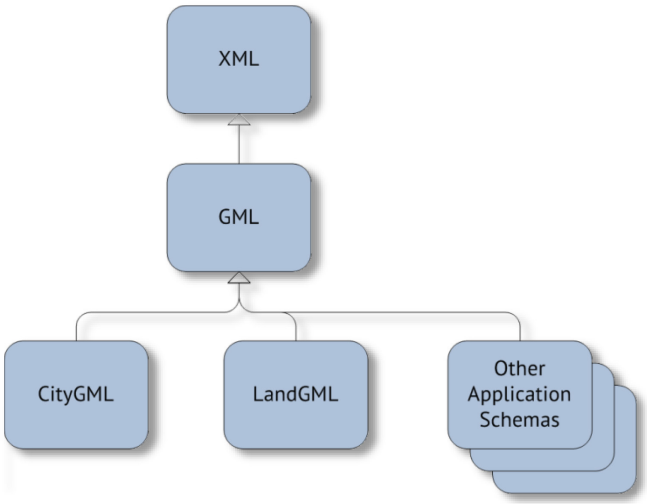


Figure 1.2: CityGML extension diagram (snowflakesoftware.com, 2014).

In order to describe a city model, CityGML utilizes thematic modules. The base module is the CityGML Core which gets extended by defined thematic modules such as Building, Bridge, Vegetation, Transportation, WaterBody, Relief etc (Table 1.1).

The file type of CityGML is either ".xml" or ".gml". Being an XML file it can be validated against the syntax and the contents. For the syntax validation a large number of open source XML editors can be used. For the content validation a simple XML editor is not sufficient. The reason is the complex schema definition of CityGML. Each thematic module is validated against the corresponding schema file. The base schema file for the core module is the CityGML.xsd which imports the schemas for the extending thematic modules, i.e. building.xsd, waterBody.xsd, relief.xsd, vegetation.zsd etc. The imported schemas import other schemas, for example gml.xsd. CityDoctor and FME are specialized software tools that can validate the contents of CityGML. For a successful validation, the user needs to know the correct CityGML schema version to use, i.e. version 0.4.0, version 1.0.0 and version 2.0.0.

CityGML module	Prefix
Appearance	app
Bridge	brid
Building	bldg
CityFurniture	frn
CityGML Core	core
CityObjectGroup	grp
Generics	gen
LandUse	luse
Relief	dem
TexturedSurface	tex
Transportation	tran
Tunnel	tun
Vegetation	veg
WaterBody	wtr

Table 1.1: CityGML modules and their prefixes. (OGC, 2012)

Buildings are defined in the Building thematic module. Building geometry can be stored in five consecutive levels of detail (LOD0 – LOD1). LOD0 represents the footprint of the buildings. LOD1 is a simple extrusion of LOD0. LOD2 introduces roof structure (Figure 1.3). LOD3 adds wall details like windows and doors. Finally, LOD4 adds interior structures in a building.

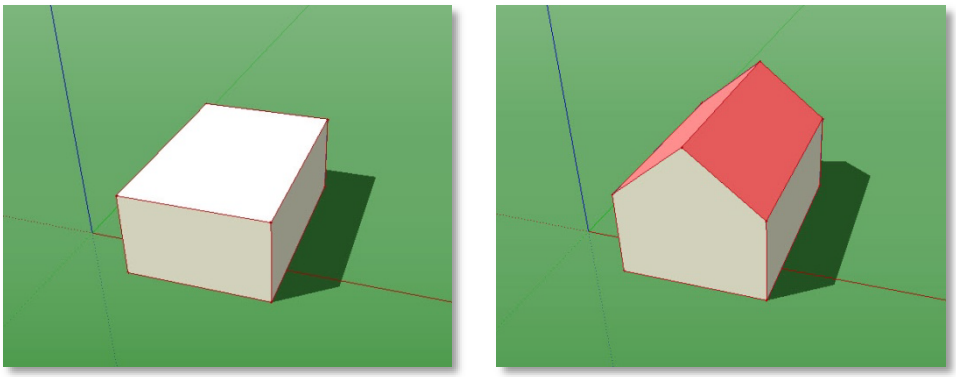
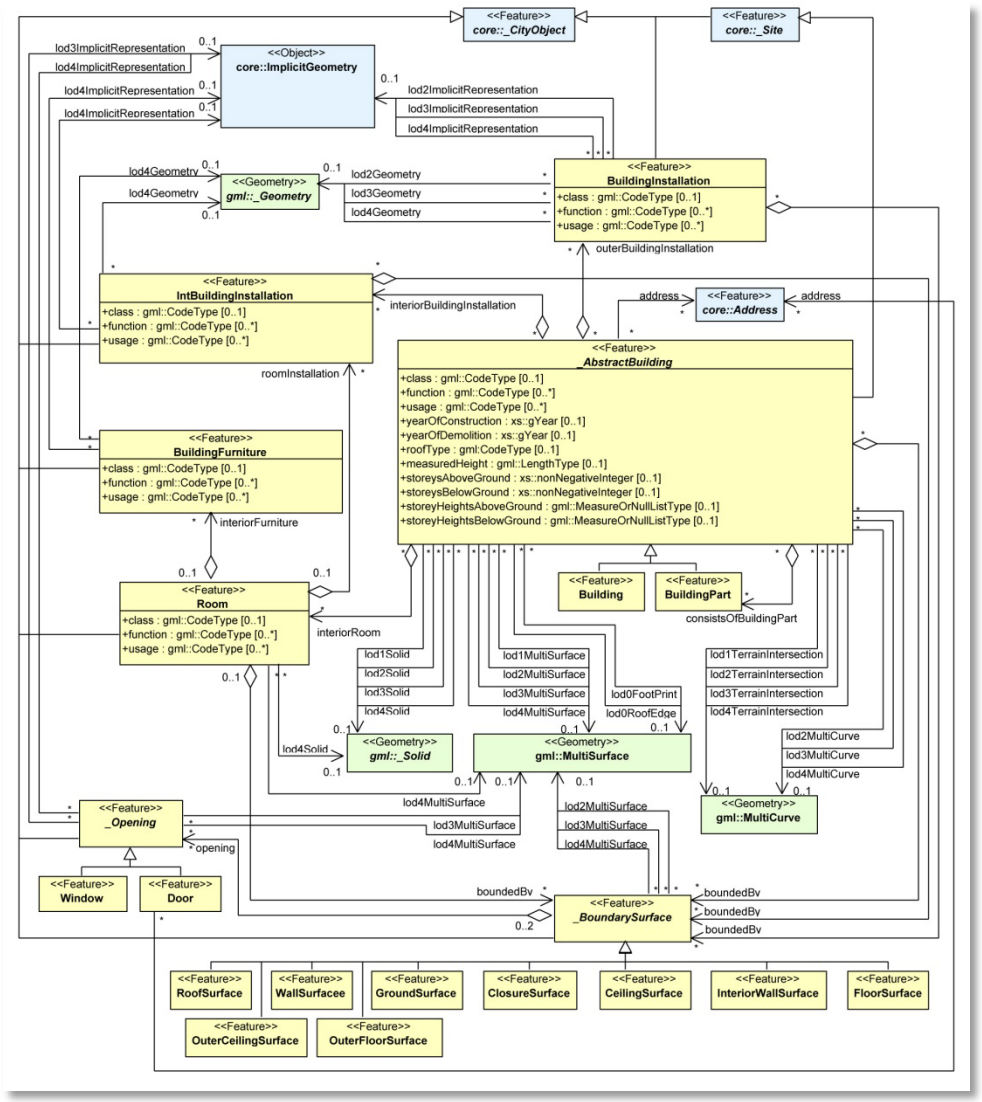


Figure 1.3: Example of multiple LODs. (a) LOD1. (b) LOD2.

The building model defines the geometric and semantic properties of a building in CityGML (UML 1.1). Buildings and building parts can be modeled as Solids (B-rep solids, Figure 1.4) or MultiSurfaces. A building can expose solid and MultiSurface geometry by using xlinks (xlinks are defined in GML standard). The solid geometry elements will reference the IDs of the surface members of the MultiSurface geometry (Figure 1.5).



UML 1.1: CityGML's building model (OGC, 2012, p. 63).

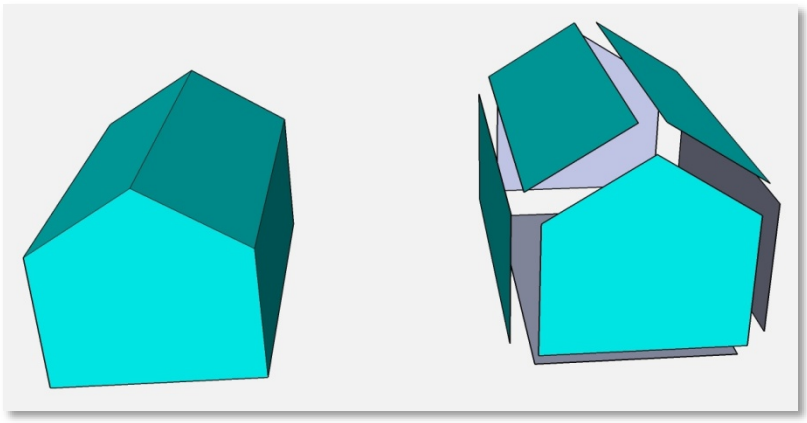


Figure 1.4: B-rep Solid example. In a B-rep a solid body is represented by planar faces. The faces are located only at the boundary of the body and enclose the body completely.

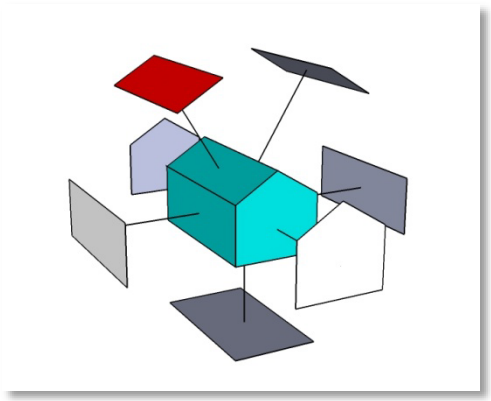


Figure 1.5: Solid definition using xlinks.

The corresponding semantic objects for a MultiSurface geometry in LOD2 are: RoofSurface, WallSurface, GroundSurface, and ClosureSurface (Figure 1.6).

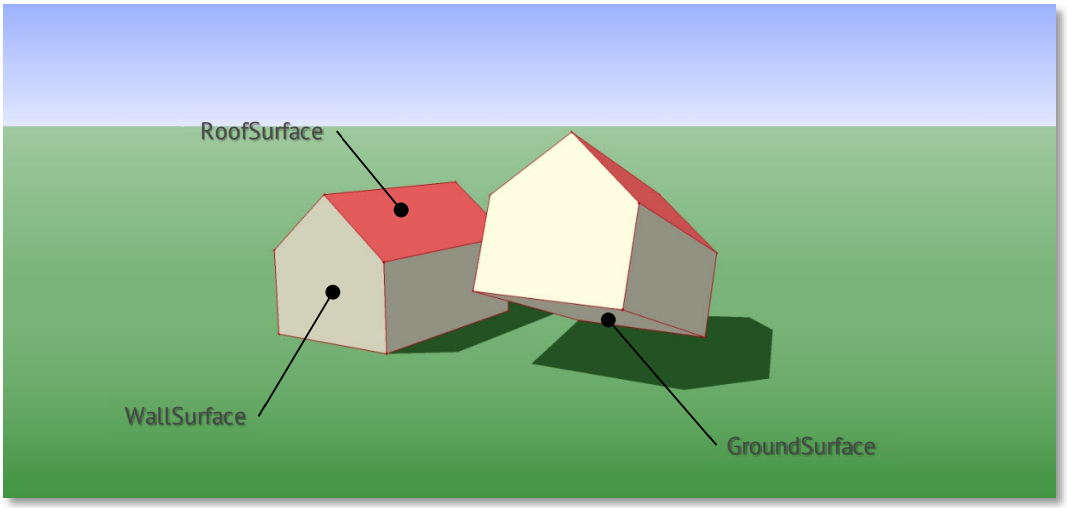


Figure 1.6: Semantic objects for Boundary Surfaces.

Geometry is modeled using surfaces. A surface can be modeled as a composite collection or an aggregate collection. A CompositeSurafce must be topologically connected. This restriction does not apply to the aggregate collections such as MultiSurface and MultiSolid.

CityGML also defines attributes for every city object. The standard defines two types of attributes: Schema attributes and generic attributes.

Schema attributes are part of the schema definition of each thematic module. The building schema defines the attributes: function, yearOfConstruction, yearOfDemolition, measuredHeight, storeysAboveGround etc.

Generic attributes are defined by the user to store additional information in a city object. There is no restriction in the number or the names of the generic attributes as long as they conform to the XML syntactic rules.

2 Research Statement

A schema valid CityGML file may contain geometric and thematic errors that do not comply with CityGML conformation requirements. In addition, different rules are applied in every CityGML dataset that characterize the city objects as valid or non-valid. It is important to study the CityGML validation tools capabilities and to define an evaluation framework in order to examine the level of concurrence between them. It is also essential to investigate if the validation process of CityGML can be characterized as "software free".

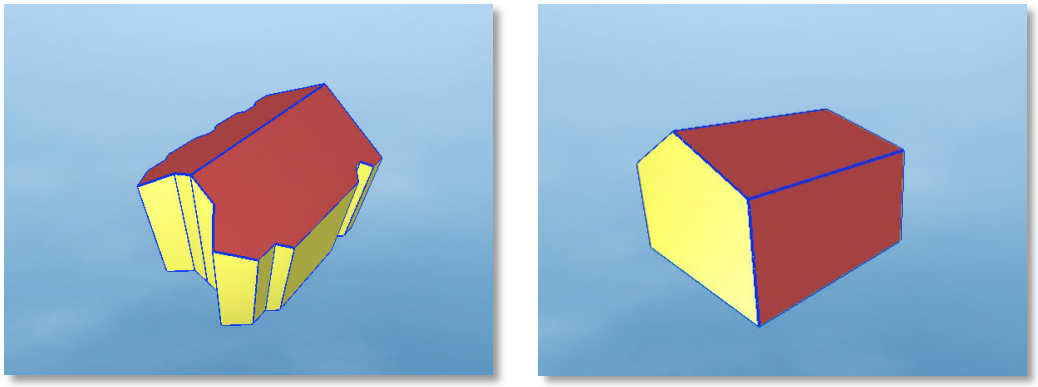
2.1 Justification

According to Blaauboer (Blaauboer et al., 2012), automatic processes can generate from LOD0 to LOD2 cityObjects. Automated techniques in 3D City Models generation introduce geometric and semantic errors in the models entities. Figure 2.1 shows some typical examples of geometric errors.

(a) (b)

Figure 2.1: Buildings with geometry errors. (a) The normal vectors of the polygons point inside the building. (b) Missing polygons.

CityGML requires a building to expose additionally Solid geometry when it exposes only MultiSurface and at the same time can form a Solid. Examples of semantic errors can be seen in Figure 2.2.



(a) (b)

Figure 2.2: Examples of semantic errors. (a) A building can form a lod2Solid but exposes only lod2MultiSurface. (b) A wall surface member is grouped in the RoofSurface.

2.2 Objectives

The main objective of the thesis is to define an evaluation framework for CityGML validation tools, CityDoctor and FME. Substantive objective is to investigate if validation of 3D city models can be characterized as software independent.

Integration of CityDoctor validation libraries in CityServer3D will extend the capabilities and allow CityServer to apply quality assessment in CityGML datasets.

3 CityDoctor And FME Evaluation

I consider needful a succinct description of the programs being evaluated before I outline the followed methodology. CityDoctor and FME can run in all platforms –Windows, Mac, Linux– and both use the latest technology in programming languages.

3.1 CityDoctor

CityDoctor is a software application for validating and healing buildings in 3D CityGML datasets, available as a desktop application. Developed by Hochschule für Technik Stuttgart, CityDoctor is written in Java. It uses the Java OpenGL library to support 3D graphics display and XML APIs such as SAX, StAx and JAXB for parsing and serialization.

The native file format of CityDoctor is .gml and .xml, both for reading and writing. It is available for German and English languages. The source code uses subversion (subversion.tigris.org) as version control system and it is divided into three branches: CityDoctorCore, CityDoctorValidationTool and

CityDoctorHealingTool. The validation tool extends the core branch and the healing tool the validation tool respectively.

The CityDoctorCore includes the basic GUI, data structures, and parsers. The CityDoctorValidationTool defines the check classes and the error classes to be generated. The CityDoctorHealingTool is responsible to try to heal the reported errors.

The check modules are divided into 4 categories: Polygon, Solid, Semantic, and Other. Polygon checks detect errors concerning vertices, intersection, and planarity issues (Table 3.1). Solid checks detect errors on solid geometry (Table 3.2). Semantic checks deal with semantic errors detected in CityGML (Table 3.3). Other checks detect structural issues in a CityGML file (Table 3.4).

Check Name	Check ID	Description	Image
Number of Points	CP_NUMPOINTS	A linear ring must consist of a minimum of 4 points.	
Closeness	CP_CLOSE	First and last point of a linear ring are identical.	$R = (P_0, P_1, P_2, P_3)$
No Duplicate Points	CP_DUPPOINT	Two edges can intersect only in one start/endpoint. Other points of intersection or touching are not allowed (to account for rounding errors or polygons which are not perfectly planar, a small tolerance $\varepsilon \in \Re$ is allowed).	$R = (P_0, P_1, P_4, P_3, P_2, P_4, P_0)$
No Self Intersection	CP_SELFINT	Two edges can intersect only in one start-/end point. Other points of intersection or touching are not allowed.	$R = (P_0, P_1, P_3, P_2, P_0)$

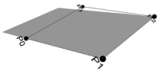
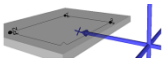
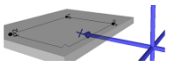
Check Name	Check ID	Description	Image
Planar Distance	CP_PLANDIST	Three points of a linear ring R are describing a plane with normal vector n. All other points must be situated in E (small deviations $\varepsilon \in \Re$ are allowed). Folds and sharp bends cannot be detected reliably.	
Planar Distance All Planes	CP_PLANDISTALL	All possible combinations of three linear independent points of a linear ring R are describing a plane with normal vector n. All other points must be situated in E (small deviations $\varepsilon \in \Re$ are allowed). In contrast to CP-PLANDIST, folds and sharp bends are detected.	
Planar Triangulation	CP_PLANTRI	The polygon P is correctly triangulated to a set of triangles (i.e. the orientation remains consistent) and the unit normal vectors for all triangles are determined. The scalar product of two normals must be less than a tolerance value ε. Folds and sharp bends are detected. Less problems with long triangles because the normal vector does not change whereas point distance is increasing compared to shorter triangles.	
Adjusted Plane	CP_PLANADJUST	An adjustment plane E with normal vector n is computed for all points of a linear ring R. All points must be situated in E (small deviations $\varepsilon \in \Re$ are allowed). Folds and sharp bends cannot be detected reliably.	

Table 3.1: CityDoctor polygon checks.

Check Name	Check ID	Description	Image
Number of Faces	CS_NUMFACES	The minimum number n of polygons to define a solid is four. They must be situated in different planes.	
No Self Intersection	CS_SELFINT	The intersection of two polygons of a solid either contains a common edge, a common point of a linear ring, or is empty. Common edges and points must be elements of both polygons.	
Edge Bounds One Polygon	CS_OUTEREDGE	Each edge of a linear ring defining a polygon is used exactly once as edge by that linear ring which defines a neighboring polygon.	
Edge Bounds More Than Two Polygons	CS_OVERUSEDEDGE	Each edge of a linear ring defining a polygon is used exactly once as edge by that linear ring which defines a neighboring polygon.	
Face Orientation Consistent	CS_FACEORIENT	Consistent orientation of polygons of a solid such that common edges according to CS- 2POLYPEREDGE are used in opposite direction.	
Normal Points Outside	CS_FACEOUT	The normal vectors of the polygons must point towards the outside of the solid.	
Connected Component	CS_CONCOMP	In the dual graph $G_S = (V_P, EP)$ of a Solid S, there exists a connected path which contains all vertices (representing the polygons of S).	
Umbrella	CS_UMBRELLA	Building is not a 3D-Manofold	
Exposed Geometry	CS_ISLOD	A building does not expose the expected geometry.	

Table 3.2: CityDoctor solid checks.

Check Name	Check ID	Description	Image
LOD1 as Solid	SEM_LOD1_ASSOLID	Checks if LOD1 models have both Solid and MultiSurface or not.	
LOD1 Building parts	SEM_LOD1_BUILDPARTS1		
LOD1 Building parts	SEM_LOD1_BUILDPARTS2		
LOD1 Building parts	SEM_LOD1_BUILDPARTS3		
LOD1 Number of Floors	SEM_LOD1_NUMFLOORS	Checks if number of floors specified in the model matches the height of building or not.	
LOD2 Ground	SEM_LOD2_GROUND	Checks if the normal direction of a polygon labeled as GroundSurface points downwards.	
LOD2 Roof	SEM_LOD2_ROOF	Checks if the normal direction of a polygon labeled as RoofSurface points upwards.	
LOD2 Wall	SEM_LOD2_WALL	Checks if the normal direction of a polygon labeled as WallSurface are horizontal or not.	
Lowest Eaves Point	SEM_LOWESTEAVESPOINT	Checks whether the height of the lowest point of all roof surfaces corresponds with the corresponding attribute value.	
Surface Area	SEM_SURFACEAREA	Checks whether the computed area of a surface corresponds to the value of corresponding attribute	
GML ID	SEM_GMLID_EXIST	Check whether the GML ID exists and is unique.	

Check Name	Check ID	Description	Image
Mandatory Attributes	SEM_MANDATORY_ATTRS	Checks if mandatory and redundant attributes exist.	
Attribute Domain	SEM_ATTRIBUTE_DOMAIN	Checks if the value of an attribute belongs in a defined domain.	

Table 3.3: CityDoctor semantic checks.

Check Name	Check ID	Description	Image
MultiSurface If Solid	C_MSIFSOLID	Checks if a Multisurface building can form a Solid.	
Buildings And BuildingPart If Solid	C_BNBPIFSOLID	Checks if polygons of building and buildingPart jointly pass all the solid tests when they are modeled as MultiSurface.	
Boundary Surface Planar Patch	C_BS_PLANAR_PATCH	Checks if a boundary surface has more than one polygon. It has to be combined with the Coplanar Surfaces Check.	
Coplanar Surfaces	C_BS_COPLANAR_SURF	Checks if two polygons are coplanar and connected.	

Table 3.4: CityDoctor other checks.

A number of check modules are dependent by other check modules. In order to perform a specific check, the dependent checks need to be performed and to return a status of “no error”. If the dependent checks return a status of “error”, they need to be healed first; otherwise the first check cannot be preformed. Table 3.5 and Table 3.6 show the dependent and independent checks.

Check ID
CP_NUMPOINTS
CP_CLOSE
CP_DUPPOINT
CS_NUMFACES
CS_2POLYPEREDGE
CS_CONCOMP

Table 3.5: CityDoctor independent checks.

Check ID	Check ID it depends on
CP_SELFINT	CP_DUPPOINT
CP_PLANDIST	CP_NUMPOINTS and CP_DUPPOINT
CP_PLANDISTALL	CP_NUMPOINTS and CP_DUPPOINT
CP_PLANTRI	CP_NUMPOINTS and CP_DUPPOINT
CS_SELFINT	CP_PLANDIST or CP_PLANDISTALL or CP_PLANTRI
CS_FACEORIENT	CS_2POLYPEREDGE
CS_FACEOUT	CS_FACEORIENT
CS_UMBRELLA	CS_2POLYPEREDGE

Table 3.6: CityDoctor dependent checks.

The graphical interface displays a list of all the buildings inside a CityGML dataset (Figure 3.1). When a building gets selected, a tabbed control displays the polygon, edges, vertices, attributes, and building errors in each tab respectively. 3D graphics are displayed on a GLCanvas. A second tabbed control displays status messages and details about the errors.

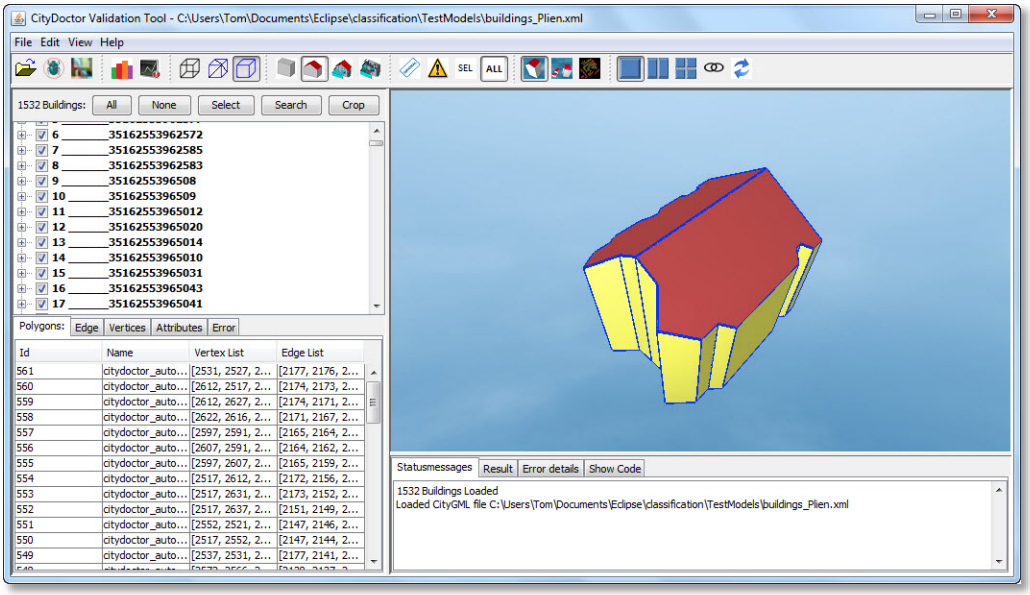


Figure 3.1: CityDoctor GUI.

CityDoctor uses the Eclipse IDE for Java for development. Eclipse was created to provide a platform on which different tools could be integrated. Eclipse IDE Framework holds a tiny core which is not a plug-in whereas the rest of the functionality in achieved by plug-ins. The Workbench and the Package Explorer are examples of the Eclipse IDE plug-ins.

> ## Quotation
>
> "Eclipse is an integrated development environment (IDE) written primarily in Java. It supports an army of programming languages, including Java, Java Platform, Enterprise Edition (Java EE), AspectJ, C/C++, Ruby, Perl, COBOL, and many others."
>
> (Silva, 2009, p. 1)

3.2 FME

FME (Feature Manipulation Engine) is a software application used to manipulate various data formats, available as a desktop and server application. Underneath both versions is the FME platform which powers the flow of spatial/non-spatial data. Developed by Safe Software, it supports complex data models, vector, raster, spatial, non-spatial, 2D, 3D, Point Clouds etc.

The first version of FME was released in 1993. Since then, a new version was released every year with additional capabilities and faster conversions. The latest version released in 2014 with additional support on cloud applications and web services. It supports Windows, Linux, and Mac Operating Systems and 32 bit/64 bit processor architectures.

Terminology

FME: Is a spatial/non-spatial data transformation platform that helps organizations more easily overcome a range of data interoperability challenges.

(safe.com, 2014a)

FME Workbench is the graphical user environment to control key functions including: Data conversion, data sharing, data transformation, data validation, data integration, and data authoring. FME Workbench can save time for users that need fast solutions avoiding menial tasks. Data Inspector is another side tool that assists to figure out problems with the data or the individual steps in a Workbench project.

Terminology

FME Workbench: The key FME Desktop application is FME Workbench, an intuitive point and click interface for graphically defining translations and transformations as a flow of data.

(safe.com, 2014a)

The main objective of FME is to deliver interoperability between formats. Translation and transformation is achieved by the "building blocks" of FME Workbench, the transformers. Transformers are a key piece of FME functionality. Each transformer accepts an input and results an output. Inputs and outputs are manipulated as "attributes". Every data structure, either spatial or non-spatial, is stored as an attribute which has a name and a data type. A user is free to add its own user defined attributes. Usually in a transformer settings, the user needs to define a specific attribute name in order the

transformation to be efficient. Figure 3.2 shows a simple example of a Workbench project.

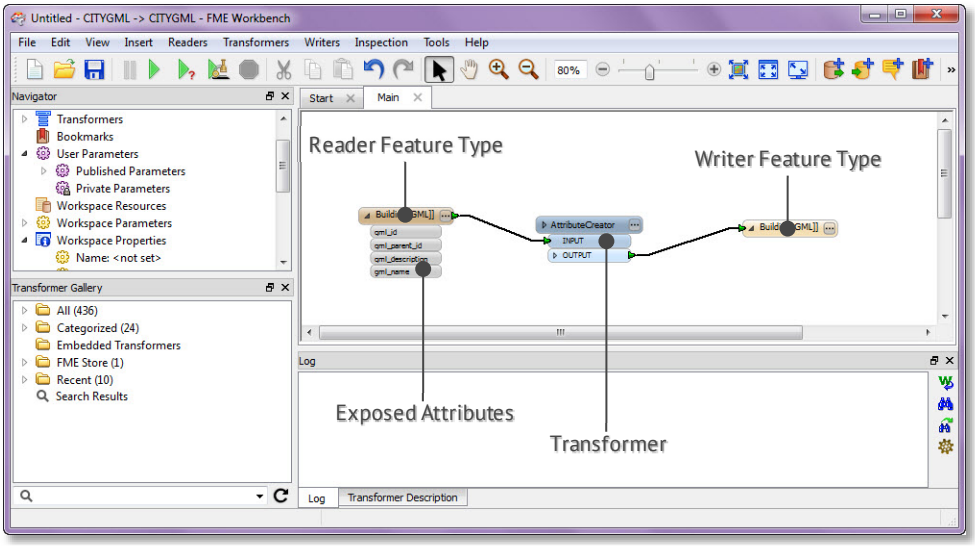


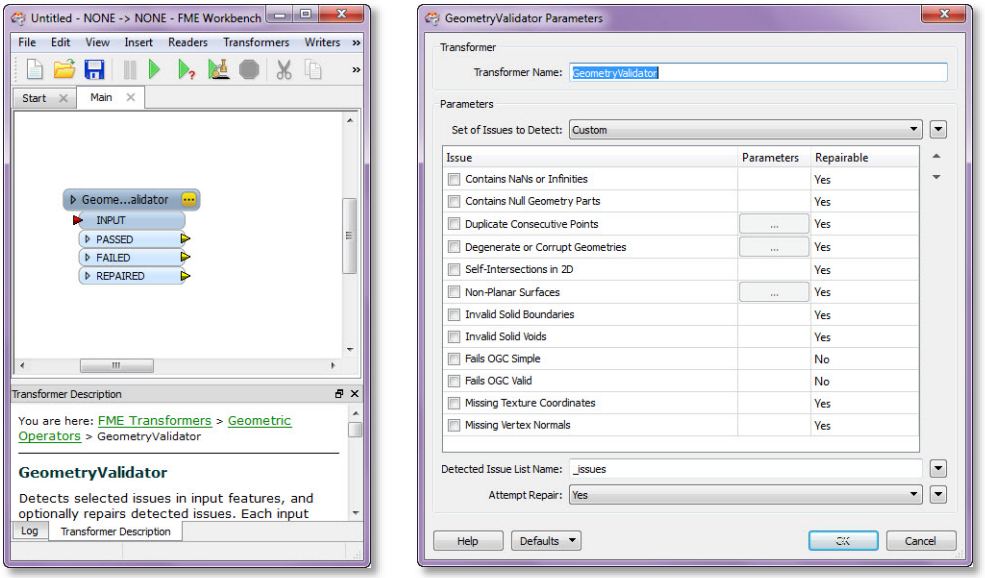
Figure 3.2: Workbench canvas components.

Terminology

FME Transformer: A transformer is an FME Workbench object that carries out the restructuring of features from the source data to the destination data. FME contains over 400 different transformers that perform different types of restructuring.

(safe.com, 2014b, p. 3)

FME can also be used for validation purposes. For geometry validation of spatial data types FME uses the Geometry Validation transformer (Figure 3.3). The input port receives the geometry content entities. Then the entities are validated against the tests available by the transformer (Figure 3.3). A geometry issue can either be repairable or not repairable (Table 3.7). If the "Attempt Repair" option is active, then the repaired entities are directed to the "REPAIRED" port. Entities that pass or fail the geometry tests are directed to the "PASSED" and "FAILED" ports respectively.



(a) (b)

Figure 3.3: FME geometry validator. (a) Input and output ports. (b) Available validation tests.

Option Name	Repairable
Contains NaNs or Infinities	Yes
Contains Null Geometry Parts	Yes
Duplicate Consecutive Points	Yes
Degenerate or Corrupt Geometries	Yes
Self-Intersections in 2D	Yes
Non-Planar Surfaces	Yes
Invalid Solid Boundaries	Yes
Invalid Solid Voids	Yes
Fails OGC Simple	No
Fails OGC Valid	No

Table 3.7: FMEs Geometry Validator options.

The input geometry can also be evaluated according to OGC standards. If an entity fails this test, an issue is generated with the message “Fails OGC Valid”. The actual geometric error can be either one or many cases shown in Table 3.8.

Error Name
Hole Outside Shell
Nested Holes
Disconnected Interior
Self Intersection
Ring Self Intersection
Nested Shells
Duplicated Rings
Too Few Points
Invalid Coordinate
Ring Not Closed

Table 3.8: Invalid OGC geometry errors.

3.3 Benchmark Definition

This section describes the research methods been used for the evaluation of CityDoctor and FME and justifies why those particular methods are the most appropriate for the evaluation. It explains how the evaluation is designed and carried out.

A primer research methodology using entities with predefined errors is not applicable. The reason is an inadequate mapping between reported errors in CityDoctor and FME. An example can be seen in Figure 3.4 where FME is reporting “OGC Fail” for a building that has a polygon with only two vertices. “OGC Fail” is a generic error that includes 10 different cases of specific errors (Table 3.8 in page 24). In addition, FME reports the location of the error in 2D coordinates, where CityDoctor reports the linear ring that contains the error, which makes difficult the correlation of the location of the errors and the evaluation outcome in general.

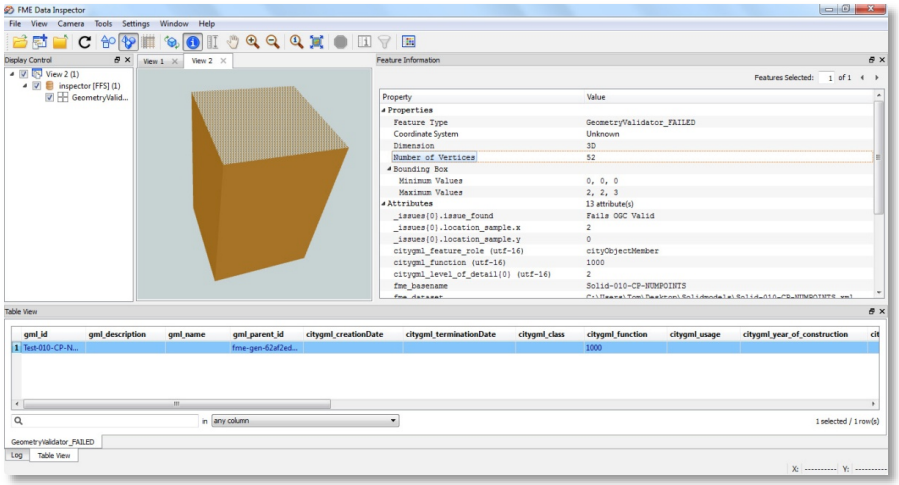


Figure 3.4: A building fails the OGC Valid rule in FME.

A case study for AdV datasets in LOD1 and LOD2 can depict graphically the areas where FME and CityDoctor concur. AdV (Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland, translated: Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany) in cooperation with the Cadastral and Surveying Authorities of the States of the Federal Republic of Germany is responsible of defining uniform regulations for the cadastre and state surveying and mapping.

AdV requires two levels of detail: LOD1 and LOD2, and a five point validation rule set. The rule set applies additional restrictions on the generic CityGML model.

The datasets provided by AdV include buildings located in Westphalia, a state northwest of Germany. In total five datasets count a sum of 4624 buildings. To make easier the reference of the datasets, aliases are used (Table 3.9).

File Name	LOD	Buildings	Location	Alias
LoD1_369_5616_1_NW.xml	LOD1	2246	Westphalia/Germany	lod1a
LoD1_453_6068_1_SH.xml	LOD1	93	Westphalia/Germany	lod1b
LoD2_369_5616_1_NW.xml	LOD2	2246	Westphalia/Germany	lod2a
4399_5352_LoD2-orig.xml	LOD2	12	Westphalia/Germany	lod2b
LoD2_523_6074_1_SH.xml	LOD2	27	Westphalia/Germany	lod2c

Table 3.9: Datasets used for the evaluation.

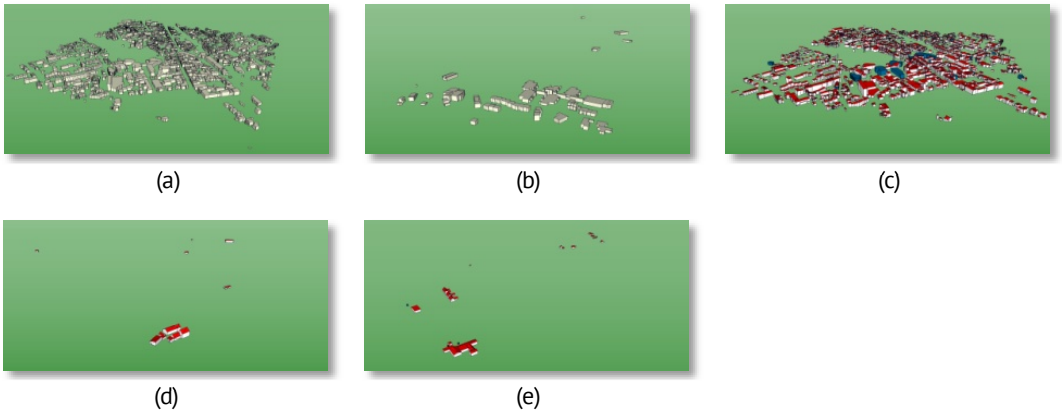


Figure 3.5: Datasets used for the evaluation. (a) lod1a. (b) lod1b. (c) lod2a. (d) lod2b. (e) lod2c.

The five point rule set provided by AdV defines a validation framework for each level of detail. Buildings records inside the German cadastre databases should comply with these rules in order to be considered as valid.

GML ID: A building/building part must have a GML id. This id has to be unique. Any missing or duplicate ids are reported as errors.

Exposed Geometry: Every building/building part in LOD1 must expose lod1Solid geometry; any other geometry is reported as error. In LOD2 every building/building part must expose a combined lod2Solid + lod2MultiSurface geometry. If a building/building part exposes only lod2MultiSurface geometry, then it is checked if it can form a solid.

Mandatory Attributes: A building/building part must store a list of attributes. These can be schema or generic attributes. Any missing attributes are reported as errors. Additional attributes that do not belong to the list are characterized as redundant and reported as errors also. The results concerning this validation point are split into two subcategories: Missing Attributes and Redundant Attributes. The list of mandatory attributes provided by AdV can be seen in Table 3.10. “Lagebezeichnung” is an optional attribute; its absence is not reported as error.

Attribute Domain: The value of an attribute must belong into a domain of values. If the value is not included in the domain, an error is reported. The domain is defined in XML format.

Valid Geometry: A building/building part must be checked for polygon, and solid errors. If a check fails, an error is reported. The semantic checks available in CityDoctor are not used, since FME does not offer corresponding semantic checks.

Name	Type	Level of detail
function	Mandatory	LOD1/LOD2
measuredHeight	Mandatory	LOD1/LOD2
DatenquelleLage	Mandatory	LOD1/LOD2
DatenquelleDachhoehe	Mandatory	LOD1/LOD2
DatenquelleBodenhoehe	Mandatory	LOD1/LOD2
BezugspunktDach	Mandatory	LOD1/LOD2
Gemeindeschluessel	Mandatory	LOD1/LOD2
roofType	Mandatory	LOD2
Lagebezeichnung	Optional	LOD1/LOD2

Table 3.10: Mandatory/Optional attributes required by AdV.

A framework wherein the validation results are gathered and stored systematically and processed requires similar structure in the data flow. Although CityDoctor offers a mechanism for reporting, the formation of this report cannot be used for direct comparison with FME validation result. For this reason, a new workspace in Eclipse with the core, validation, and healing branch of CityDoctor imported, allows to alter the source code, in order to adjust the validation report output (Figure 3.6).

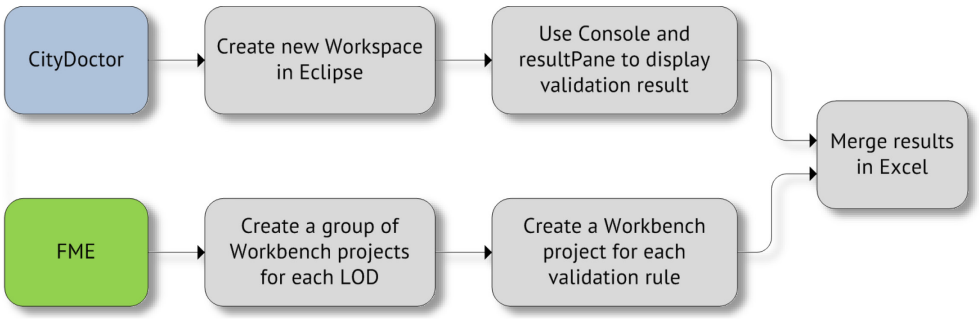


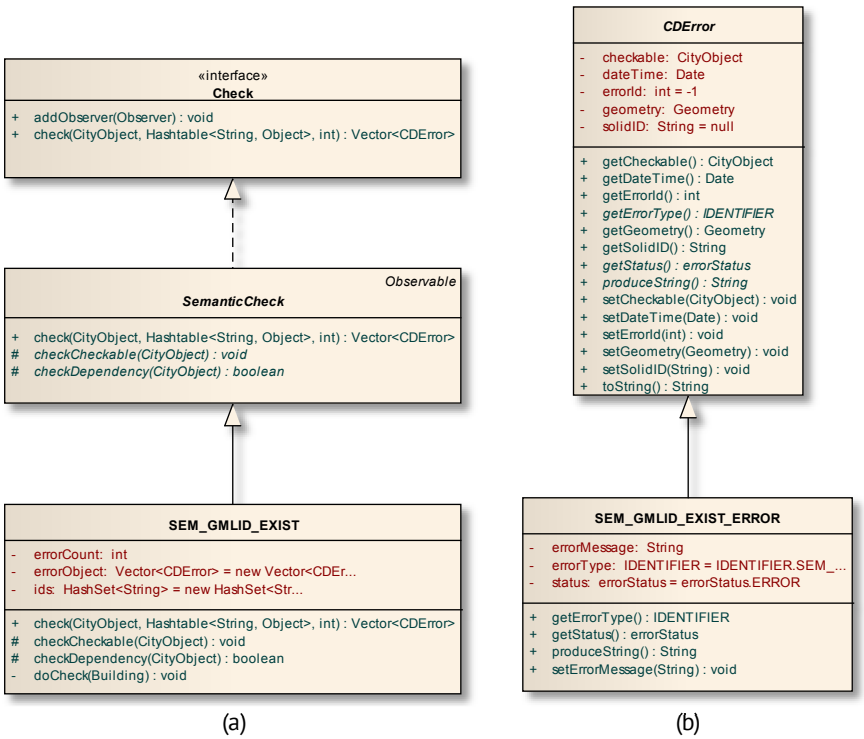
Figure 3.6: Overview of the evaluation methodology.

3.3.1 Implementations in CityDoctor

CityDoctor is designed to validate geometry and semantics of CityGML buildings. AdV 5 point rule set defines new requirements for a building which CityDoctor cannot validate. 4 of 5 validation points (GML ID, Expose Geometry, Mandatory Attributes, and Attribute Domain) need implementation in CityDoctor.

GML ID is implemented as a Semantic check (UML 3.1). The abstract class "SemanticCheck" realizes the "Check" interface. The concrete class "SEM_GMLID_EXIST" extends the abstract class and defines private and public members to check the building, generate and store the detected error.

The "SEM_GMLID_EXIST_ERROR" is the generated error which extends the abstract error class "CDError". The error object holds information about the date/time it was generated, the error id and the checked building instance.

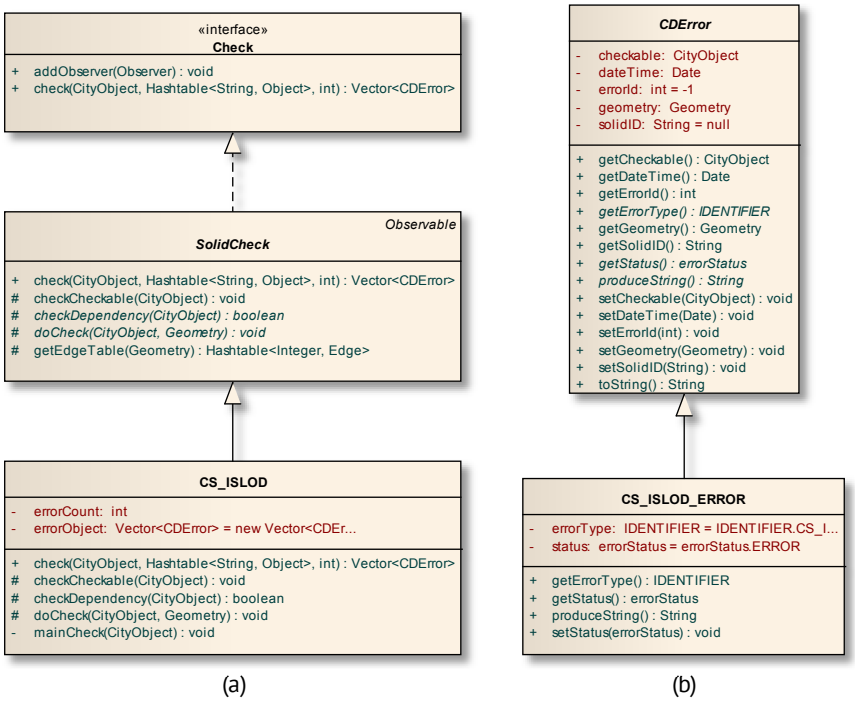


UML 3.1: GML ID implementation in CityDoctor.

Exposed Geometry is implemented as a Solid check (UML 3.2). The abstract class "SolidCheck" realizes the "Check" interface. The concrete class

"CS_ISLOD" extends the abstract class and defines private and public members to check the building, generate and store the detected error.

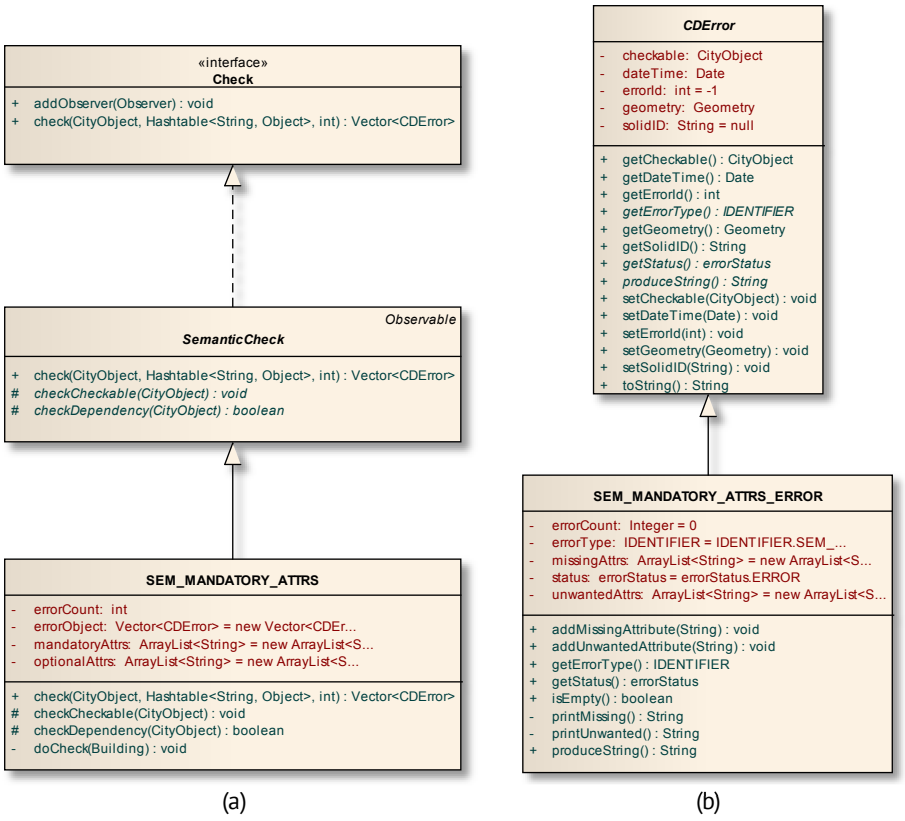
The "SEM_GMLID_EXIST_ERROR" is the generated error which extends the abstract error class "CDError". The error object holds information about the date/time it was generated, the error id and the checked building instance.



UML 3.2: Exposed Geometry implementation in CityDoctor.

Mandatory Attributes is implemented as a Semantic check (UML 3.3). The abstract class "SemanticCheck" realizes the "Check" interface. The concrete class "SEM_MANDATORY_ATTRS" extends the abstract class and defines private and public members to check the building, generate and store the detected error.

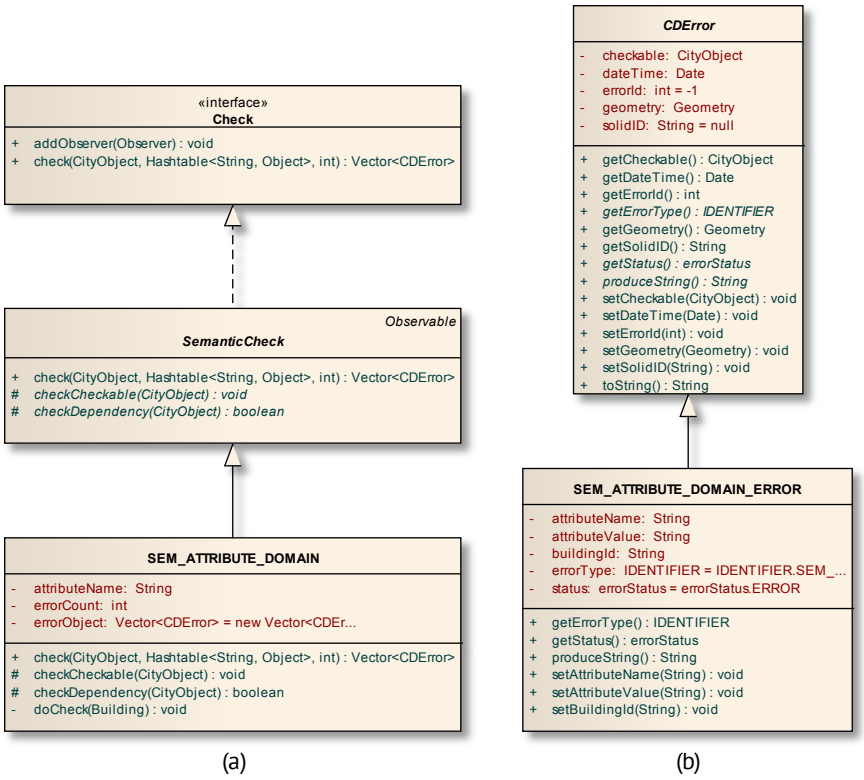
The "SEM_ MANDATORY_ATTRS_ERROR" is the generated error which extends the abstract error class "CDError". The error object holds information about the date/time it was generated, the error id and the checked building instance. It also implements methods to add missing and unwanted attributes.



UML 3.3: Mandatory Attributes implementation in CityDoctor.

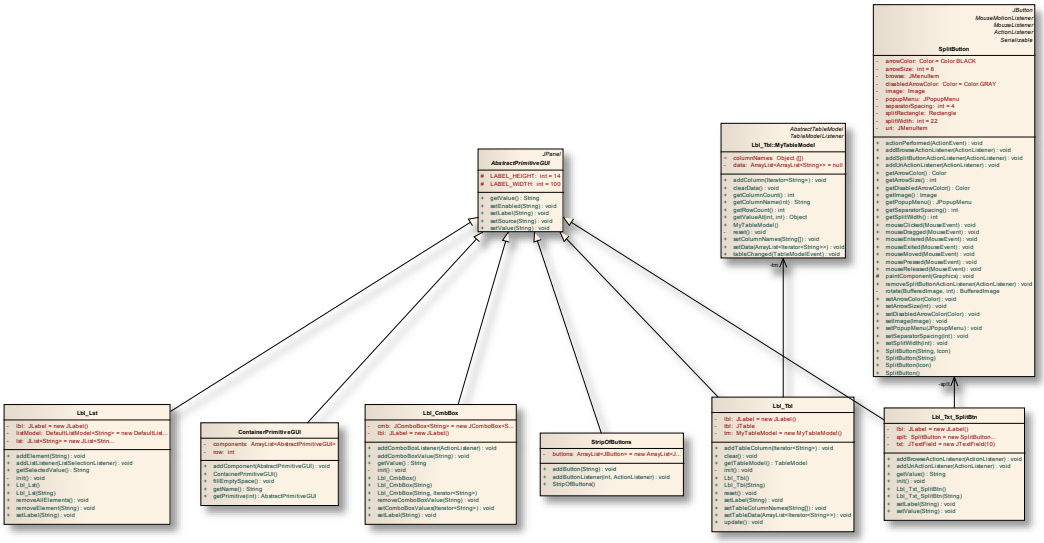
Attribute Domain is implemented as a Semantic check (UML 3.4). The abstract class “SemanticCheck” realizes the “Check” interface. The concrete class “SEM_ATTRIBUTE_DOMAIN” extends the abstract class and defines private and public members to check the building, generate and store the detected error.

The “SEM_ ATTRIBUTE_DOMAIN_ERROR” is the generated error which extends the abstract error class “CDError”. The error object holds information about the date/time it was generated, the error id and the checked building instance. It also implements methods to store the erroneous attribute name and value.



UML 3.4: Attribute Domain implementation in CityDoctor.

The Attribute Domain introduced a new challenge for CityDoctor i.e. the implementation of a new GUI for the settings of a check module. The implementation realizes the idea of reusable GUI components without hardcoded functionality. Each component is characterized as primitive and inherits from an abstract primitive (UML 3.5).



UML 3.5: GUI implementation in Attribute Domain.

An example of a concrete GUI primitive is the ComboBox Setting (Figure 3.7). It is a combination of a JPanel, a JLabel, and a ComboBox control. The rest of the controls implemented can be seen in Figure 3.8 – Figure 3.11.

Figure 3.7: ComboBox Setting.

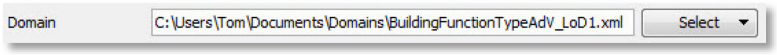


Figure 3.8: TextBox with SplitButton Setting.

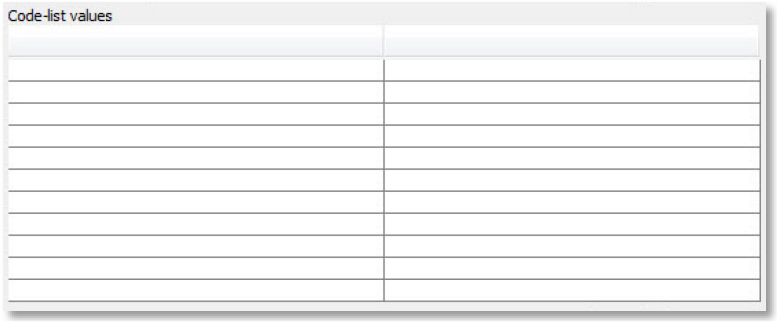


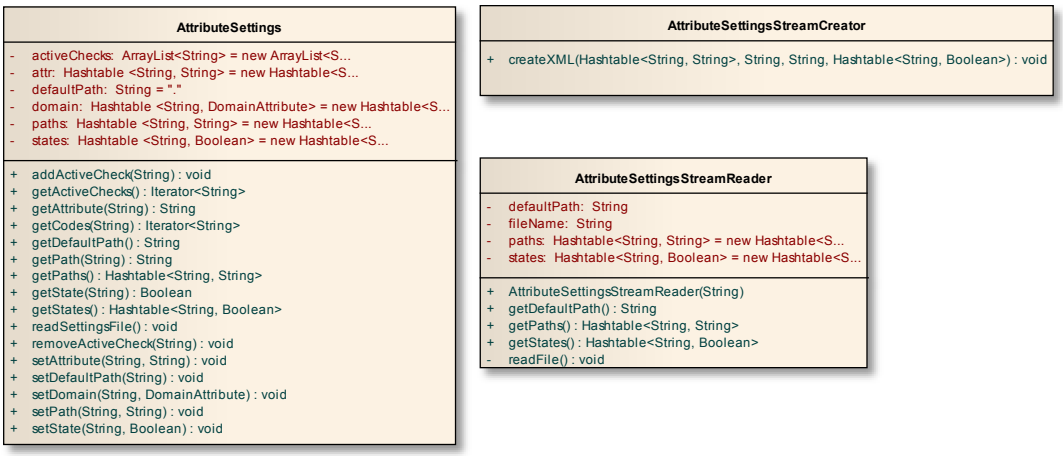
Figure 3.9: JLabel with a JTable control.

Figure 3.10: ButtonStrip.

Figure 3.11: Label with a JList.

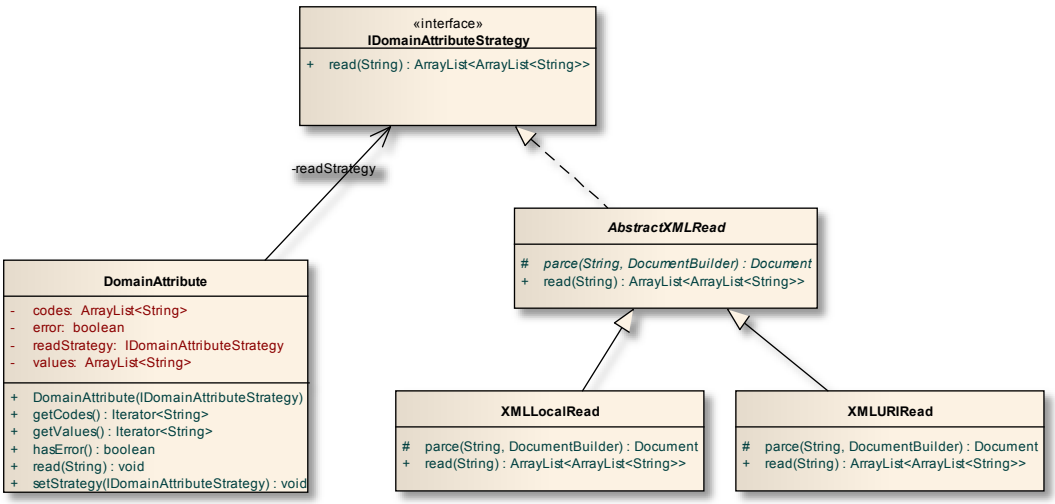
All primitives are stacked in a primitive container which uses the GridBag Layout to store the primitive controls in vertical orientation (each control below the other).

Additional classes for serialization for each domain settings and values can be seen in UML 3.6. “AttributeSettings” is the class to store the paths of the domain files, the active attributes ect. The “StreamCreator” is responsible to save the current state of the “AttributeSettings” class and the “StreamReader” is responsible to restore its state.



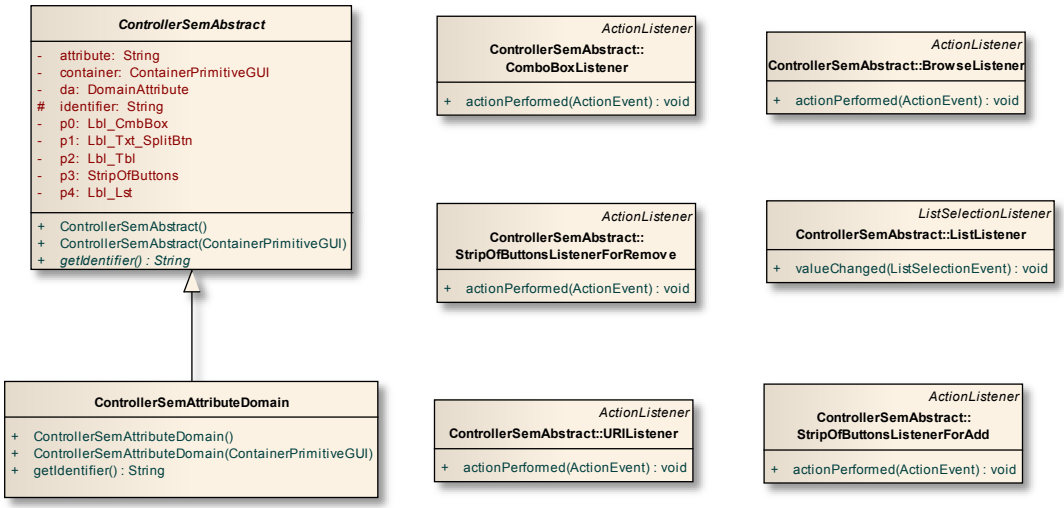
UML 3.6: Attribute Domain serialization classes.

Reading of a value domain utilizes two strategies (UML 3.7). XMLLocalRead reads a domain from the local file system whereas XMLURIRead reads a domain from a remote location. The values of the domain are stored in the "DomainAttribute" class (UML 3.7).



UML 3.7: Domain reading strategies.

The Model-View-Controller Architecture (MVC) is used to decouple the GUI from the Attribute Settings in UML 3.6. The abstract controller extends the concrete controller (UML 3.8). The event listeners are implemented as inner classes.



UML 3.8: Model-View-Controller implementation.

3.3.2 Implementations in FME

The same validation rule set needs to be implemented in FME as well. The implemented FME Workbench projects are divided into two categories: Projects for LOD1 and projects for LOD2.

All implemented Workbench projects are visualized as simplified versions of the original ones, since their extent is too large. In dark brown color are the readers feature types. The small boxes with vertical orientation show the reader type. There are cases where a CityGML dataset is read not as CityGML type but as simple XML. In light blue color are the FME transformers or a combination of numerous transformers that describe an action performed. The light brown boxes are the FME writers. In all projects the writer type is the Microsoft Excel format (.xlsx).

The GML ID validation in LOD 1 uses the CityGML reader (Figure 3.12). Buildings/ building parts are processed by conditional transformers, called “testers”. The results in Excel are written into different worksheets (an Excel worksheet is defined as a feature type for the Excel writer in FME). The “summary” sheet stores the total number of the processed and erroneous buildings. The “duplicate” sheet stores the buildings/building parts with duplicate IDs.

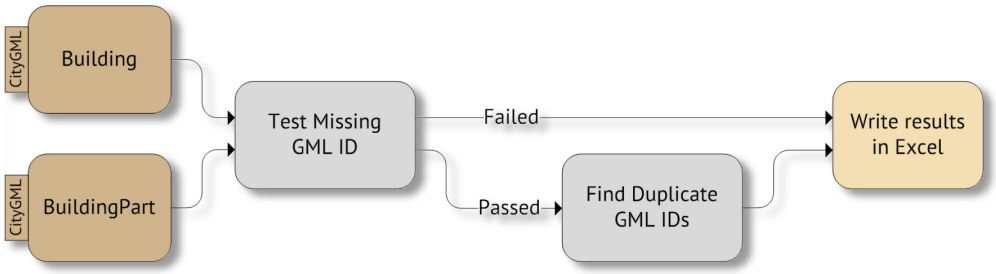


Figure 3.12: Simplified FME workflow for GML ID validation in LOD1.

The Exposed Geometry in LOD1 utilizes the CityGML feature types: Buildng, BuildingPart, GroundSurface, WallSurface, and RoofSurface (Figure 3.13). The solid geometry is exposed by the building/building part feature and the MultiSurface geometry in exposed by the BoundarySurface types. The geometry is extracted and processed.

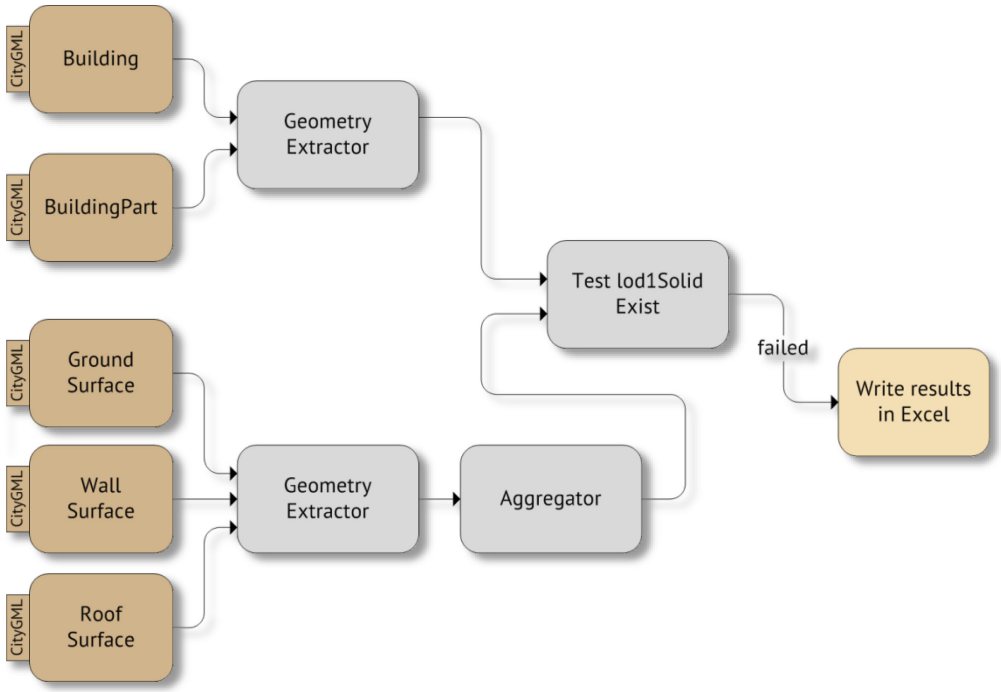


Figure 3.13: Simplified FME workflow for Exposed Geometry validation in LOD1.

The Mandatory Attribute rule in FME reads CityGML datasets as CityGML and plain XML. The plain XML reader is used because it stores the generic attributes as a list, which is a dynamic way of reading. The CityGML reader exposes all attributes (schema and generic) as single reader attributes which defines a static behavior. For example, if a building exposes an additional generic attribute, then it has to be exposed manually from the user, otherwise it

is not visible. The missing and redundant attributes are written in corresponding sheets in Excel (Figure 3.14).

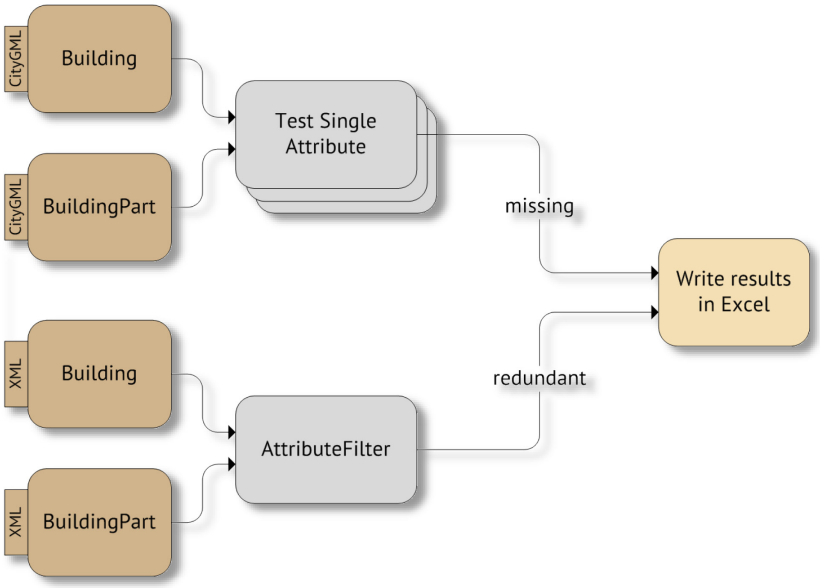


Figure 3.14: Simplified FME workflow for mandatory attributes validation in LOD1.

The Attribute Domain rule in LOD1 uses the XML reader for the domains. The values of the attributes are extracted and then tested with a feature merger transformer. The buildings/building parts IDs with erroneous or missing attribute values are written in Excel (Figure 3.15).

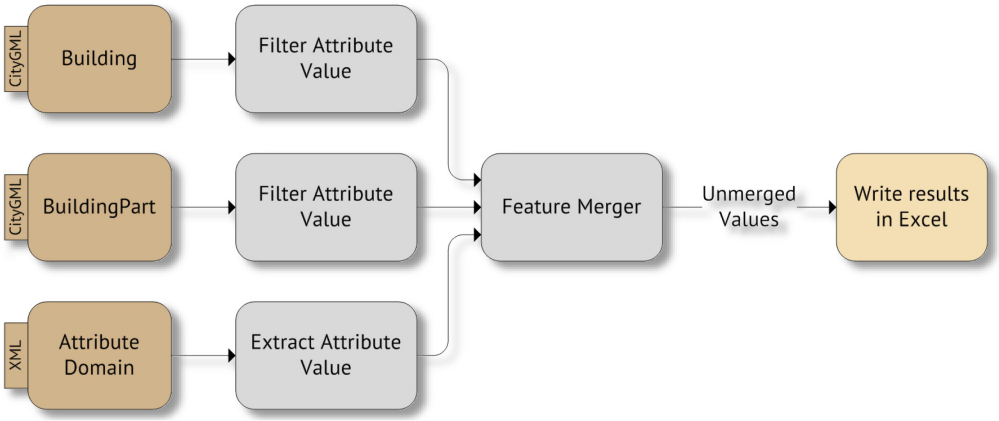


Figure 3.15: Simplified FME workflow for attribute domain validation in LOD1.

The Geometry Validation in LOD1 uses the CityGML reader. Geometry is extracted and then tested using the geometryValidator transformer. A summary with all the buildings/building parts IDs is written is Excel. The "geometry errors" sheet stores the ID and the corresponding issue (Figure 3.16).

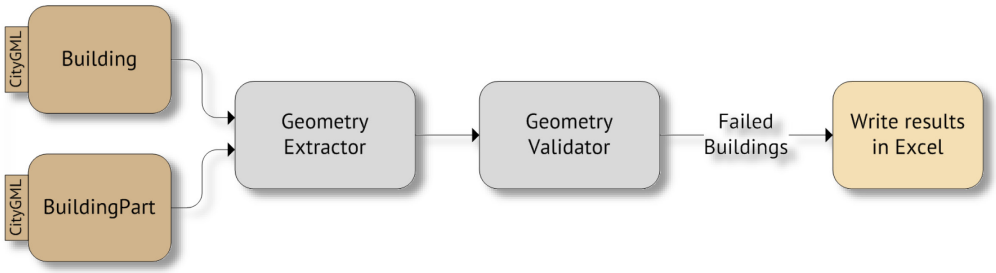


Figure 3.16: Simplified FME workflow for geometry validation in LOD1.

The second group of Workbench projects implements the AdV 5 point rule set for LOD2. In some cases the implementation is different in LOD2 since the demands are in different in this level of detail.

The GML ID validation in LOD 2 uses the CityGML reader (Figure 3.17). Buildings/ building parts are processed by conditional transformers. The results in Excel are written into different worksheets. The "summary" sheet stores the total number of the processed and erroneous buildings. The "duplicate" sheet stores the buildings/building parts with duplicate IDs.

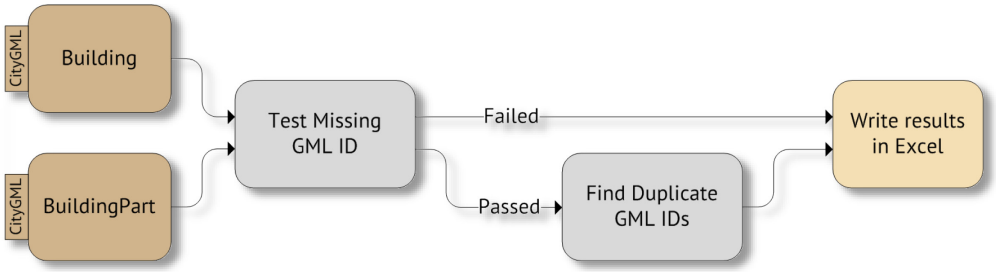


Figure 3.17: Simplified FME workflow for GML ID validation in LOD2.

The Exposed Geometry in LOD2 utilizes the CityGML feature types: Buildng, BuildingPart, GroundSurface, WallSurface, and RoofSurface. The solid geometry is exposed by the building/building part feature and the MultiSurface geometry in exposed by the BoundarySurface types. The geometry is extracted and processed. In case a building/building part exposes only lod2MultiSurface geometry, it is sent to SolidBuilder transformer in order to be tested if it can form a solid. The results are written in Excel into four worksheets: Summary, Errors, Can Form Solid, and Cannot Form Solid (Figure 3.18).

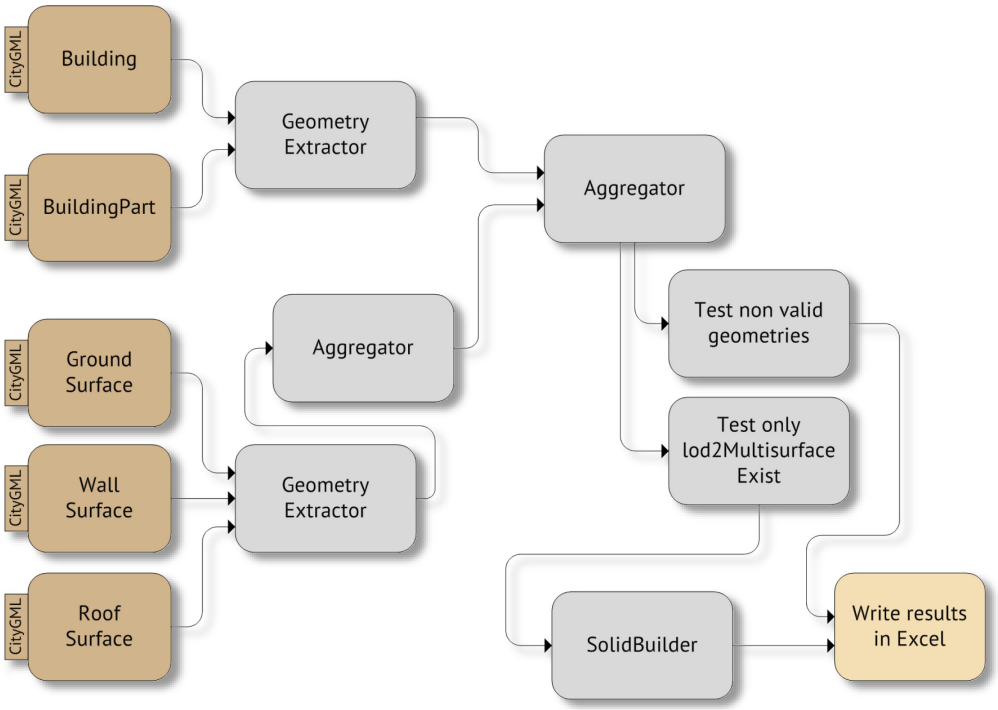


Figure 3.18: Simplified FME workflow for Exposed Geometry validation in LOD2.

The Mandatory Attribute rule in LOD2 reads CityGML datasets as CityGML and plain XML. The attribute roofType is an additional mandatory attribute. The missing and redundant attributes are written in corresponding sheets in Excel (Figure 3.19).

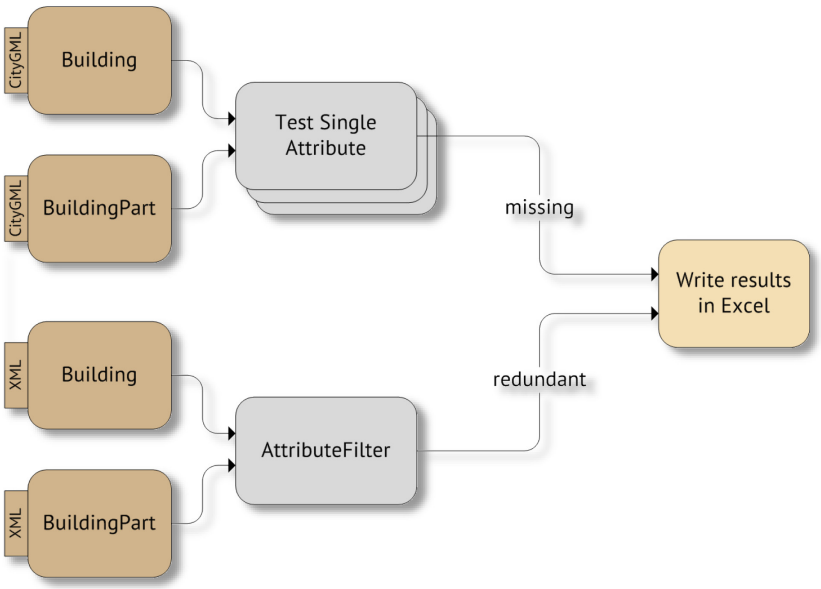


Figure 3.19: Simplified FME workflow for mandatory attributes validation in LOD2.

The Attribute Domain rule in LOD2 uses the XML reader for the domains. The values of the attributes are extracted and then tested with a feature merger transformer. The roofType is an additional attribute which domain is checked. The buildings/building parts IDs with erroneous or missing attribute values are written in Excel (Figure 3.20).

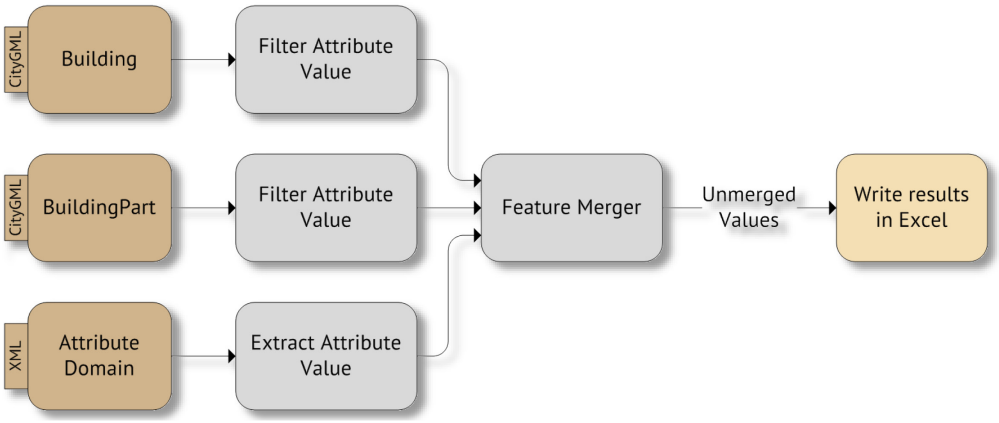


Figure 3.20: Simplified FME workflow for attribute domain validation in LOD2.

The Geometry Validation in LOD2 uses the CityGML reader. Building and BuildingPart exposes the solid geometry. The boundary surfaces: Ground Surface, Wall Surface and Roof Surface expose the MultiSurface geometry. Geometry is extracted and then tested using the geometryValidator transformer. A summary with all the buildings/building parts IDs is written is Excel. The "geometry errors" sheet stores the ID and the corresponding issue.

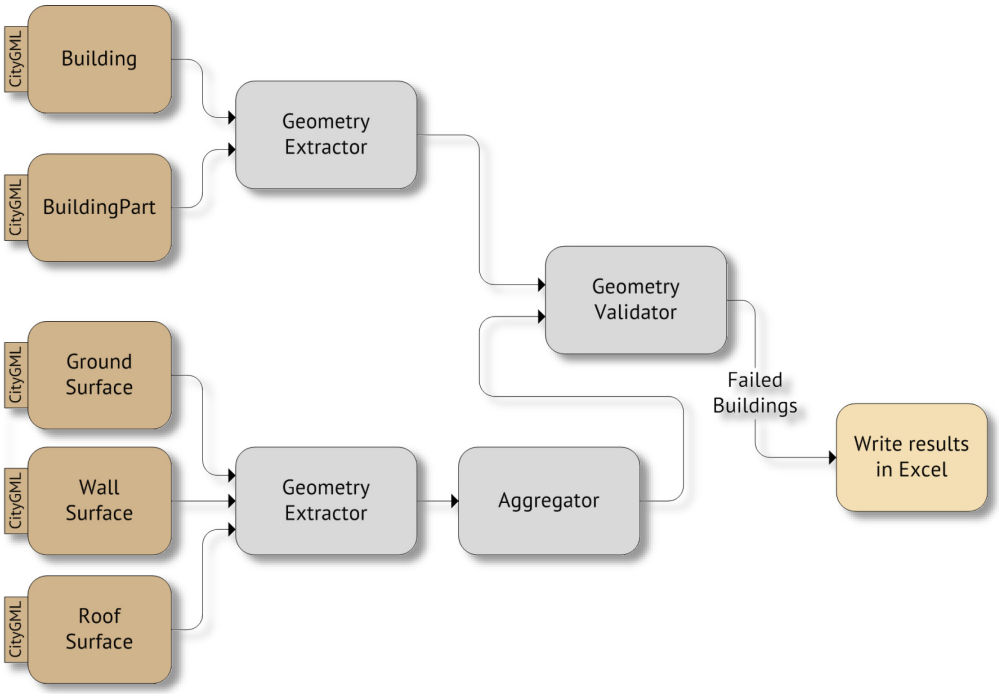


Figure 3.21: Simplified FME workflow for geometry validation in LOD2.

3.3.3 Additional tools

An additional workbench in FME that is able to extract buildings and building parts according to a list of IDs assists for further analysis on specific parts of a cityModel (Figure 3.22). The list of IDs is provided in a form of a text file. It is important to mention that building parts are elevated to buildings. Specifically, buildingPart elements are renamed to Building elements.

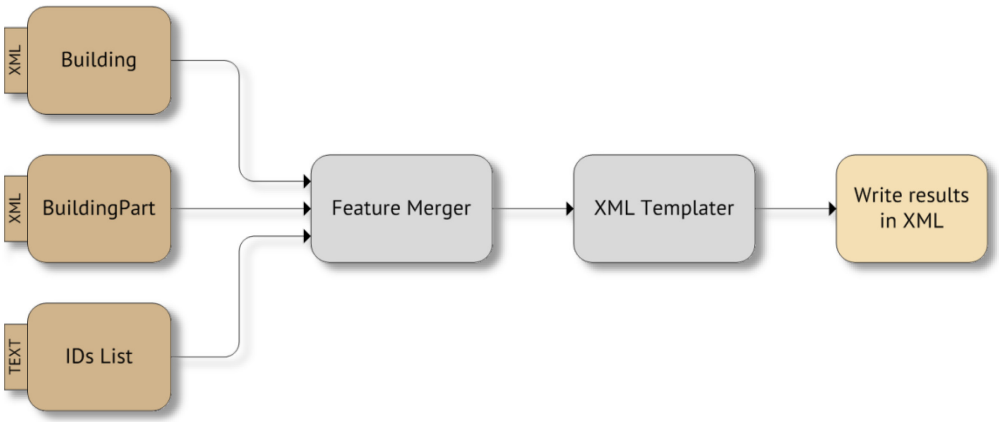


Figure 3.22: Building and BuildingPart extractor implemented in FME.

A batch CityGML schema validator accepts a list of file paths and the location of the schema. The XML Validator transformer validates the files in the list and outputs the results in the FME Inspector (Figure 3.23).

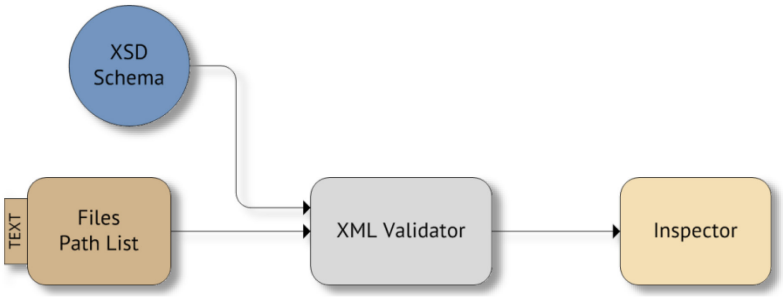


Figure 3.23: CityGML batch schema validator.

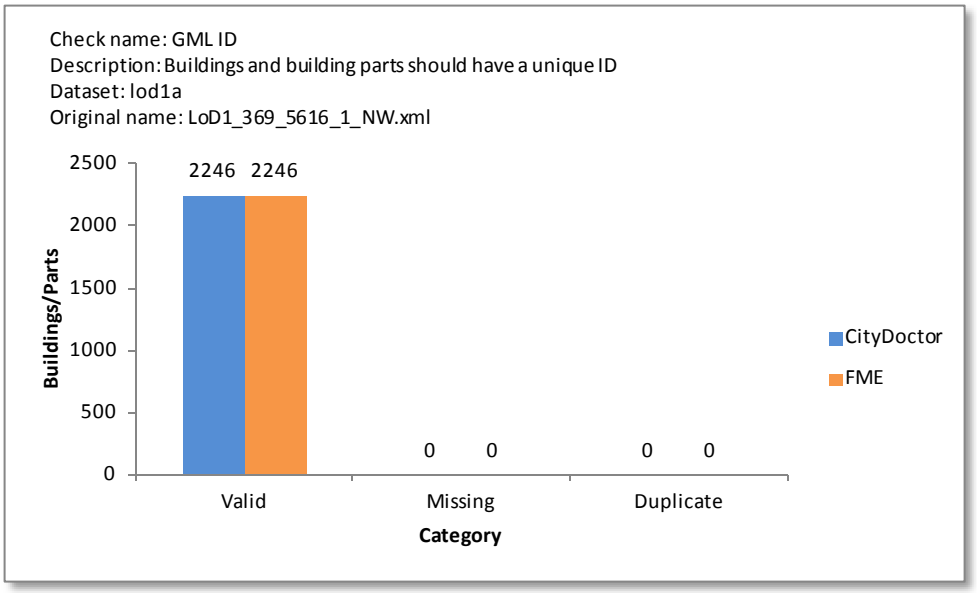
3.4 Results

The "findings" are presented as an organized summary of the information that has been collected in a form of graphs. Each graph displays metadata in the header area about the check name, description, dataset alias and original dataset name.

All datasets passed the CityGML schema validation defined by CityGML 1.0 specification. The batch schema validator, implemented in FME, was used.

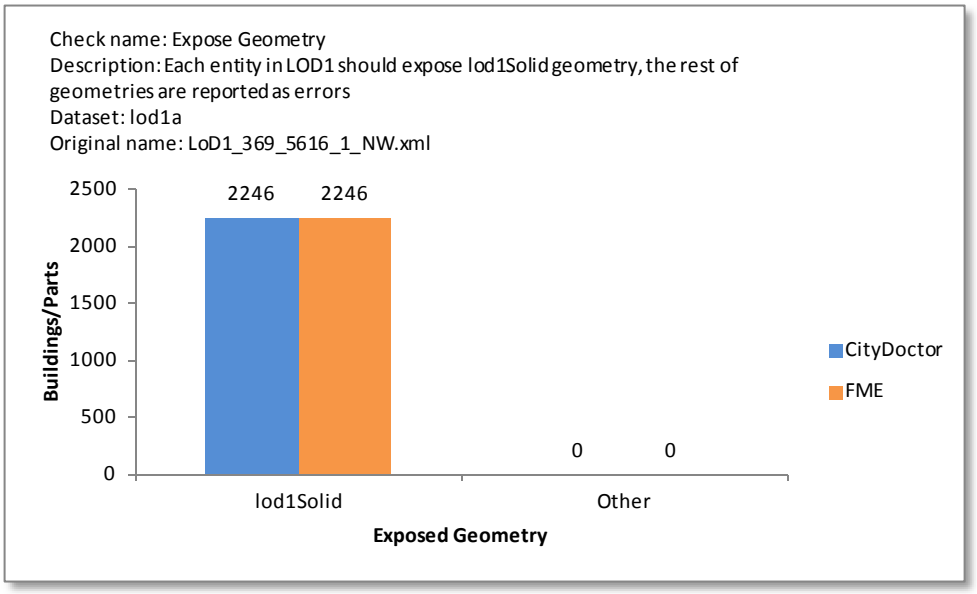
3.4.1 Results for lod1a dataset

In lod1a dataset both tools give the same results as appears in Graph 3.1. All buildings/building parts have unique IDs. None of them have missing or duplicate IDs.



Graph 3.1: GML ID validation result for lod1a dataset.

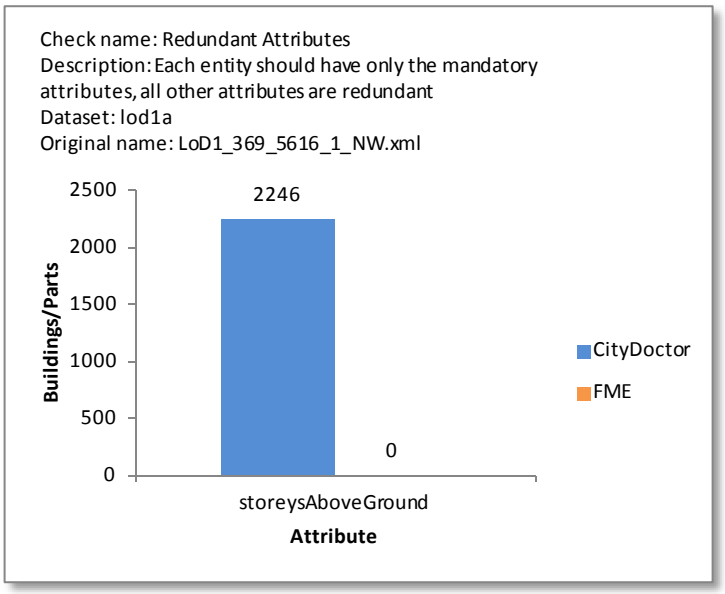
All buildings/building parts expose the expected geometry type for the level of detail they belong. Any other geometry type is not present in this dataset (Graph 3.2).



Graph 3.2: Expose Geometry validation result for lod1a dataset.

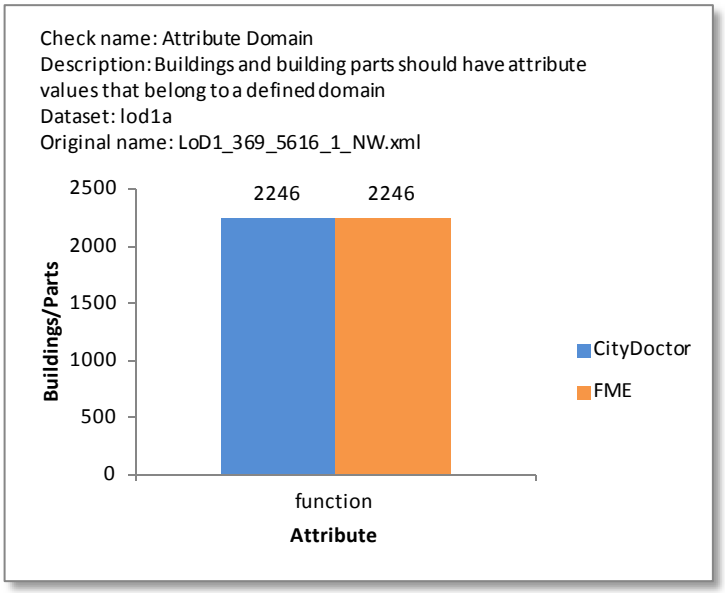
The results for the redundant attributes reveal a weak feature in FME (Graph 3.3). Since storeysAboveGround is a schema attribute, it cannot be read by the XML reader in FME. Changing the reader type from XML to CityGML

and selecting this attribute manually may solve the problem but this solution violates a basic principle that the benchmark should not adjust to the data.



Graph 3.3: Redundant attributes for lod1a dataset.

The attribute function in lod1a dataset has values that do not belong in a domain defined by AdV (Graph 3.4). All values specifically have a prefix code followed by an underscore and the actual value (Table 3.11).

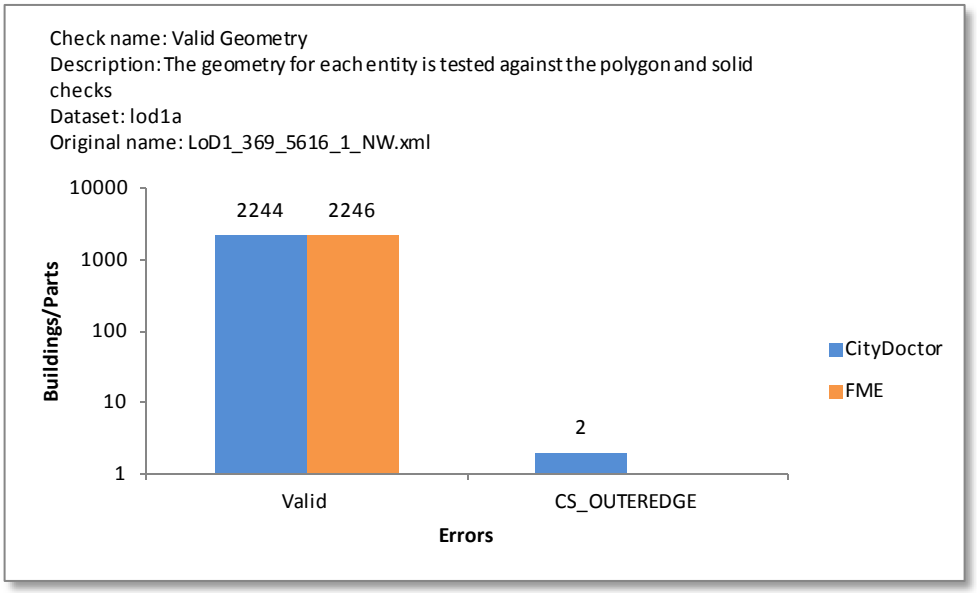


Graph 3.4: Attributes with values that do not belong to a defined domain.

GML ID	Attribute Name	Value
DENW_d9e29371-7561-4fcb-9817-a866694cff01	function	11_1003
DENW_c182a96c-eb84-452f-b09a-d7e6d3f82b6c	function	11_1003
DENW_1d59de30-2cc5-45e1-a168-e27d3cda0d1a	function	11_1003

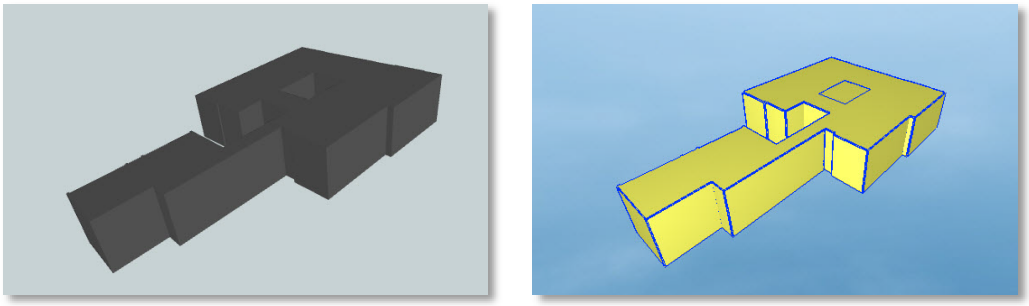
Table 3.11: Detail from the benchmark raw results shows the erroneous values of function.

The geometry test shows similar results for both tools. CityDoctor utilizes the polygon and solid checks [add ref]. FME utilizes solid and OGC valid checks [add ref]. FME reports no erroneous buildings. CityDoctor reports two CS_OUTEREDGE errors.



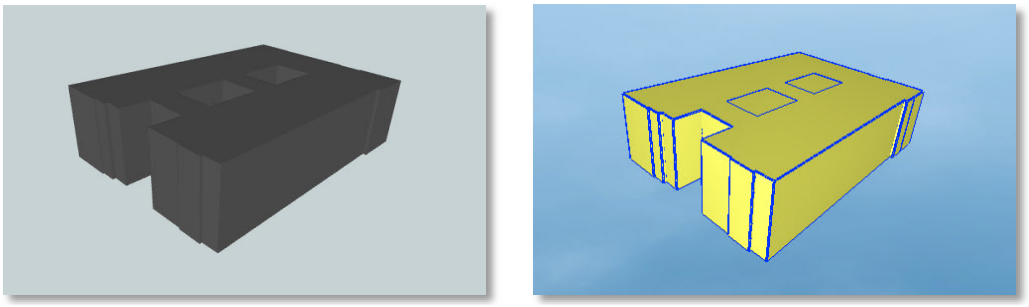
Graph 3.5: Geometry validation results for lod1a dataset.

Visualization of both erroneous buildings that CityDoctor reports shows difference in the way CityGML is parsed. CityDoctor does not fully support polygons with holes. The polygons that cover the holes report a CS_OUTEREDGE error (Figure 3.24 and Figure 3.25).



(a) (b)

Figure 3.24: Parsing CityGML gives different results. (a) FME can detect holes in polygons. (b) CityDoctor does not fully support holes in polygons.

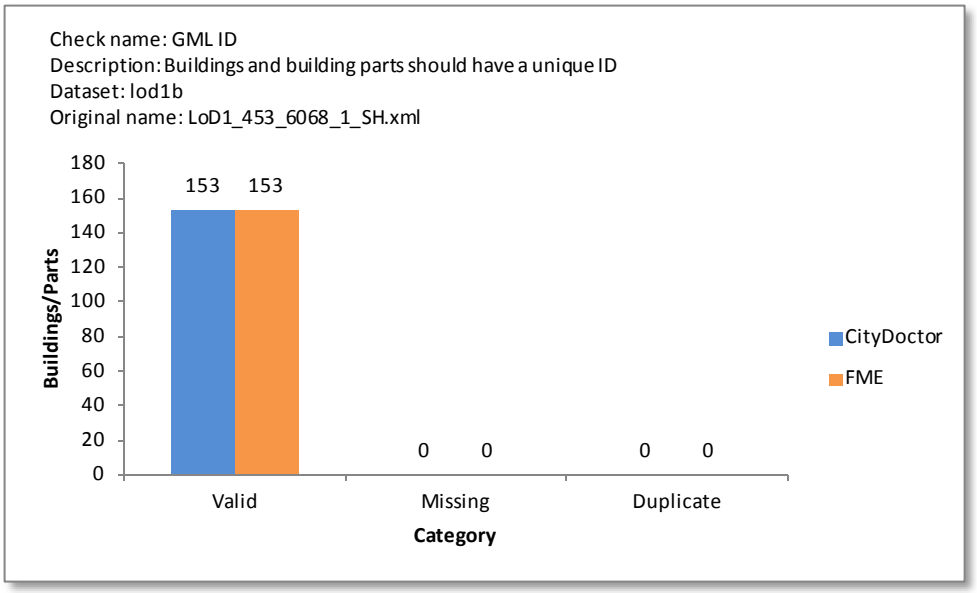


(a) (b)

Figure 3.25: The second building with two holes in a polygon. (a) FME parses the file correctly. (b) CityDoctor cannot detect the inner holes.

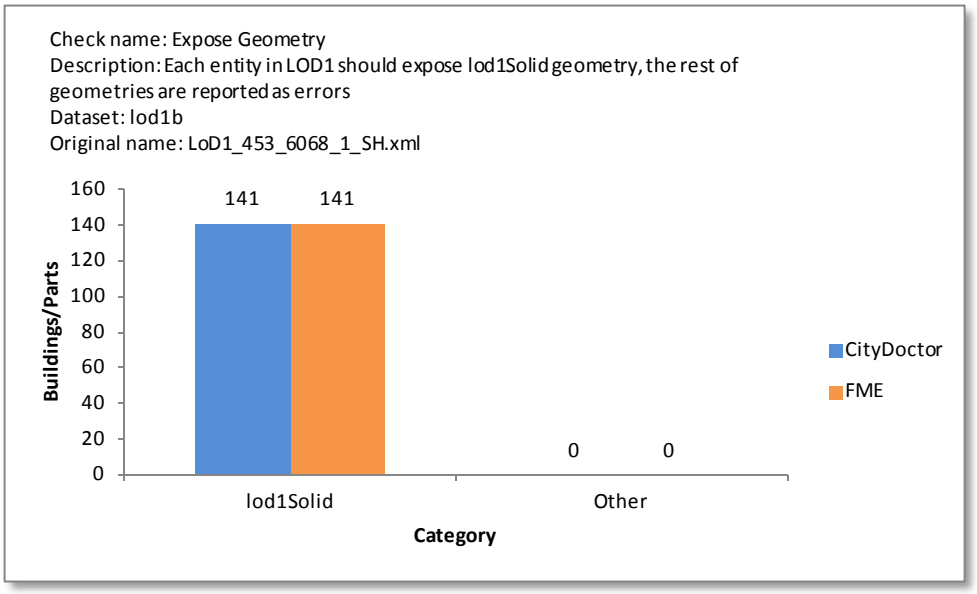
3.4.2 Results for lod1b dataset

In lod1b dataset both tools give the same results as appears in Graph 3.6. All buildings/building parts have unique IDs. None of them have missing or duplicate IDs.



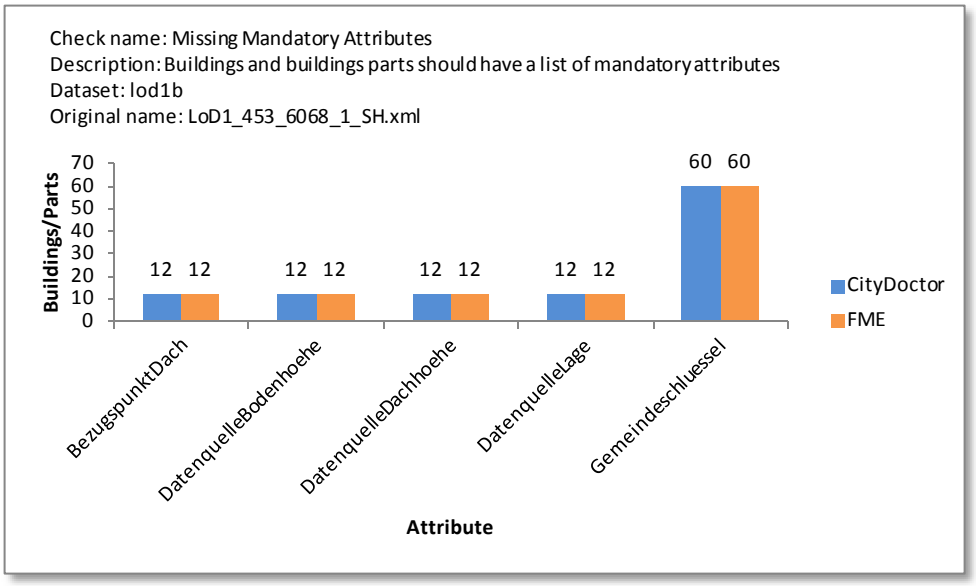
Graph 3.6: GML ID validation result for lod1b dataset.

All buildings/building parts expose the expected geometry type for the level of detail they belong. Any other geometry type is not present in this dataset (Graph 3.7).



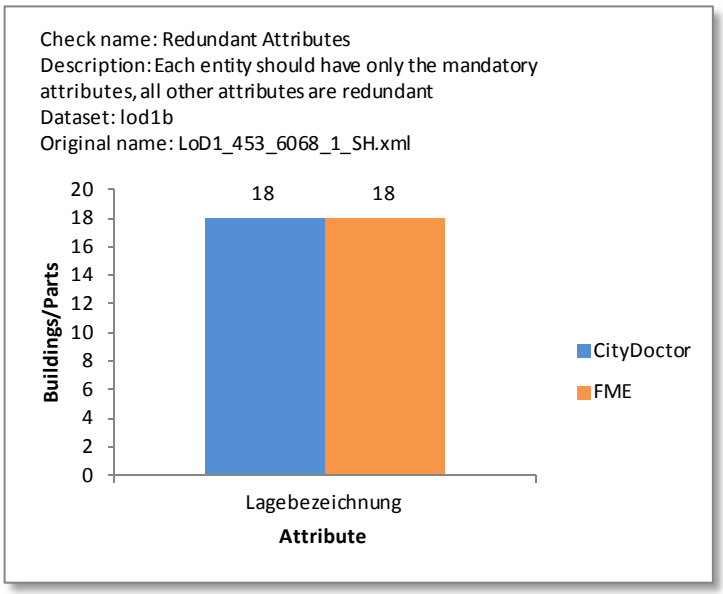
Graph 3.7: Expose Geometry validation result for lod1b dataset.

The results for the missing mandatory attributes are identical (Graph 3.8). 60 building parts report the Gemeindeschluessel attribute as missing. This attribute exists as a building attribute.



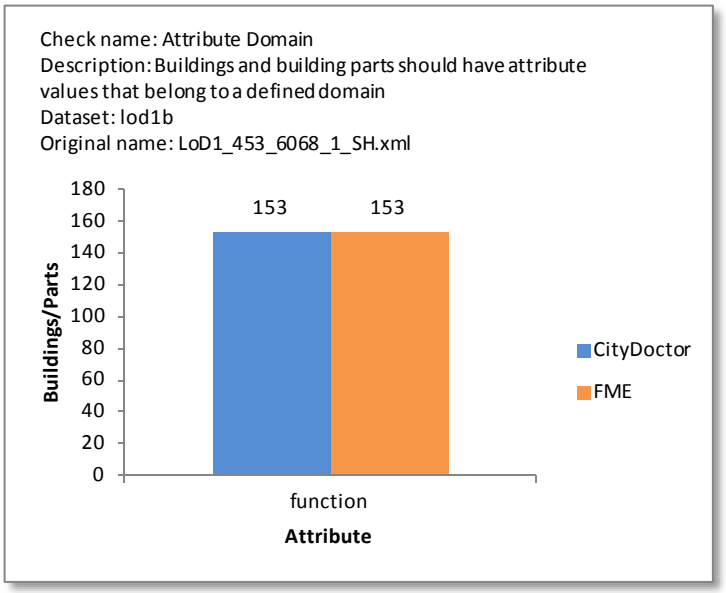
Graph 3.8: Missing attributes for lod1b dataset.

CityDoctor and FME report “Lagebezeichnung” as a redundant attribute in 18 buildings/building parts (Graph 3.9).



Graph 3.9: Redundant attributes for lod1b dataset.

The attribute function in lod1b dataset has values that do not belong in a domain defined by AdV (Graph 3.10). All values have a prefix code followed by an underscore and the actual value (Table 3.12).

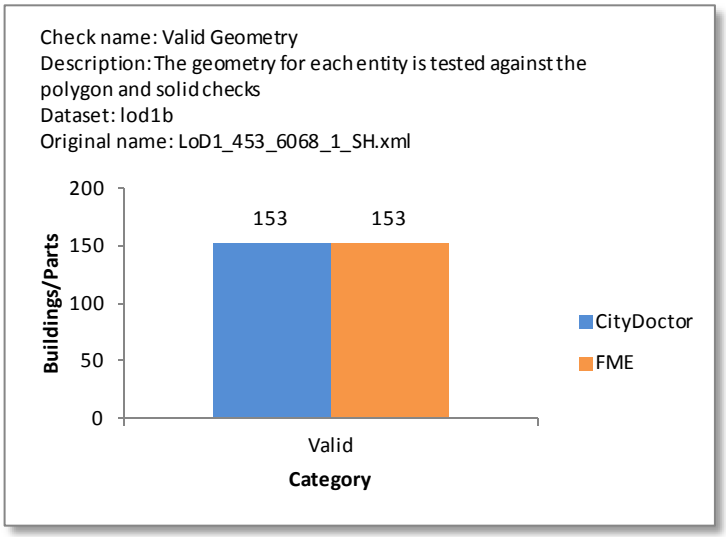


Graph 3.10: Function attribute has values that do not belong to a defined domain.

GML ID	Attribute Name	Value
DESH001e01de-0000-1000-7d5b-00000139bb62	function	31001_2143
DESH001eca1e-0000-1000-7d5b-00000139bb62	function	31001_1310
DESH001eca1f-0000-1000-7d5b-00000139bb62	function	31001_1310

Table 3.12: Detail from raw results shows the erroneous values of function.

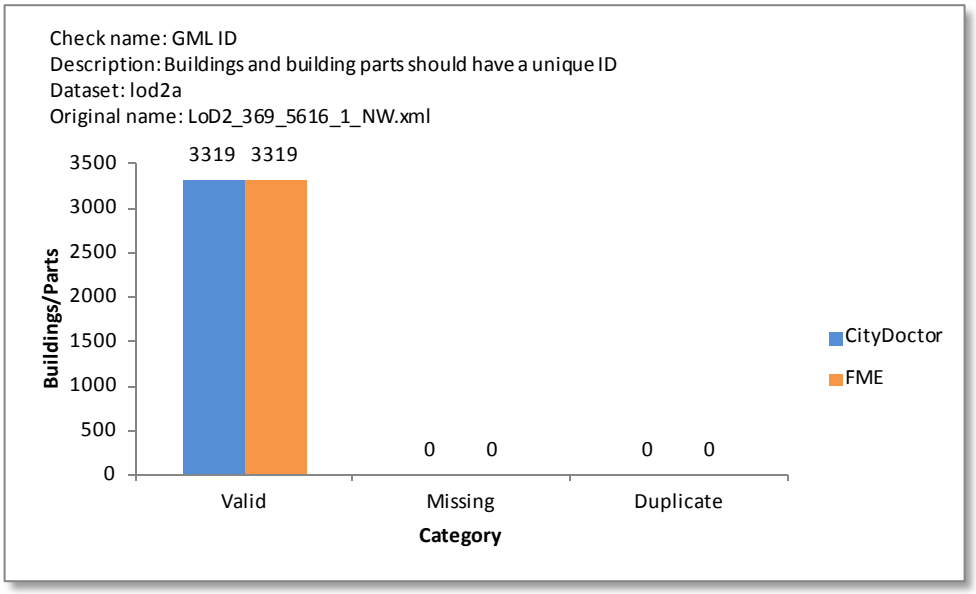
The valid geometry reports no errors in lod1b dataset. All entities have valid geometry.



Graph 3.11: Geometry validation results for lod1b dataset.

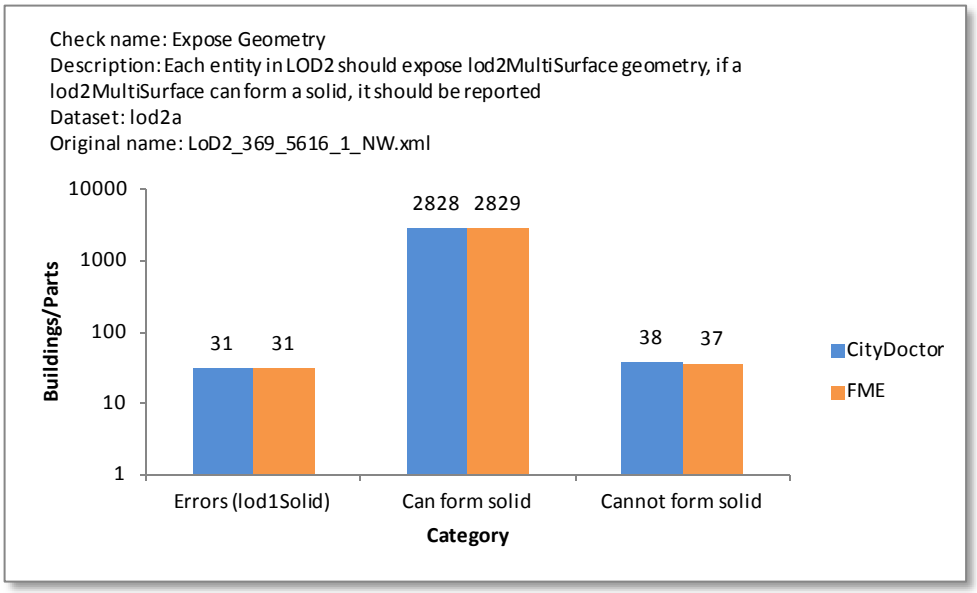
3.4.3 Results for lod2a dataset

In lod2a dataset both CityDoctor and FME give the same results as appears in (Graph 3.12). All buildings/building parts have unique IDs. None of them have missing or duplicate IDs. Some buildings in lod1a dataset are split in building parts in lod2a dataset. This is the reason more entities appear in the results.



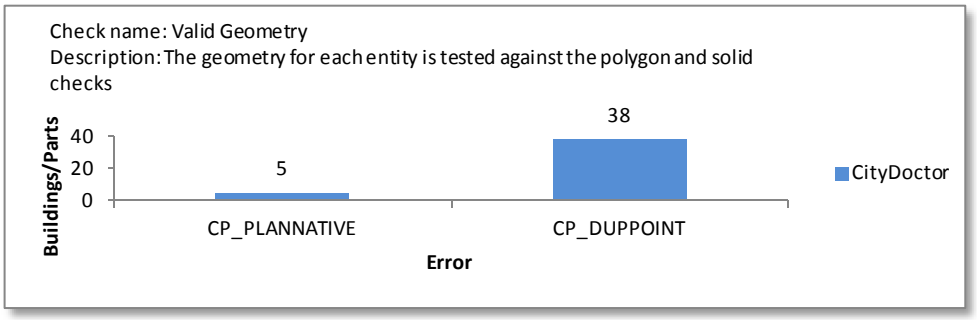
Graph 3.12: GML ID validation result for lod2a dataset.

The exposed geometry results appear to be almost the same (Graph 3.13). FME reports one additional entity that exposes lod2MultiSurface and can form solid, thus FME reports one less entity that cannot form solid. Comparison of the IDs of the entities in the "cannot form solid" category shows that 35 of them are common between both tools.

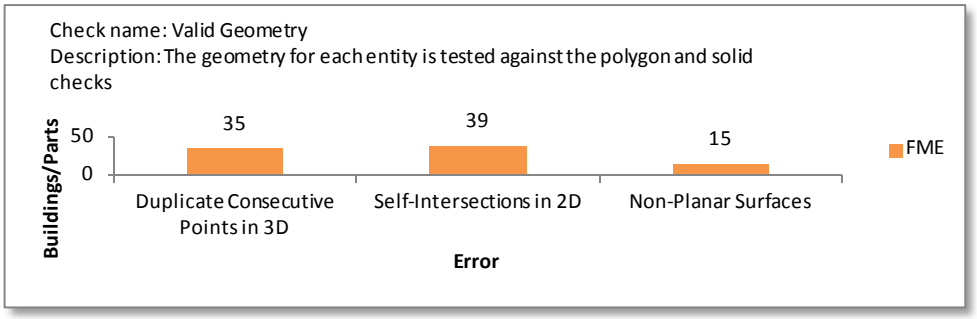


Graph 3.13: Expose Geometry validation result for lod2a dataset.

The use of the building/building part extractor, implemented in FME, to further analyze the error distribution in entities that cannot form solid geometry shows that mostly duplicate points are responsible. The rest of the errors do not seem to correlate (Figure 3.26).



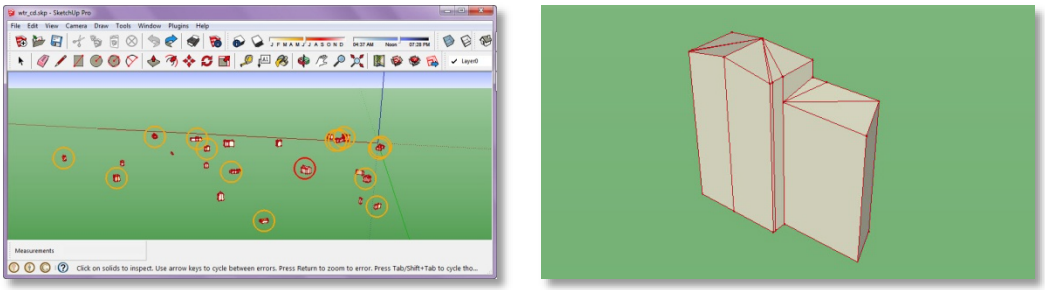
(a)



(b)

Figure 3.26: Further analysis in entities that cannot form solid geometry. (a) CityDoctor error distribution. (b) FME error distribution.

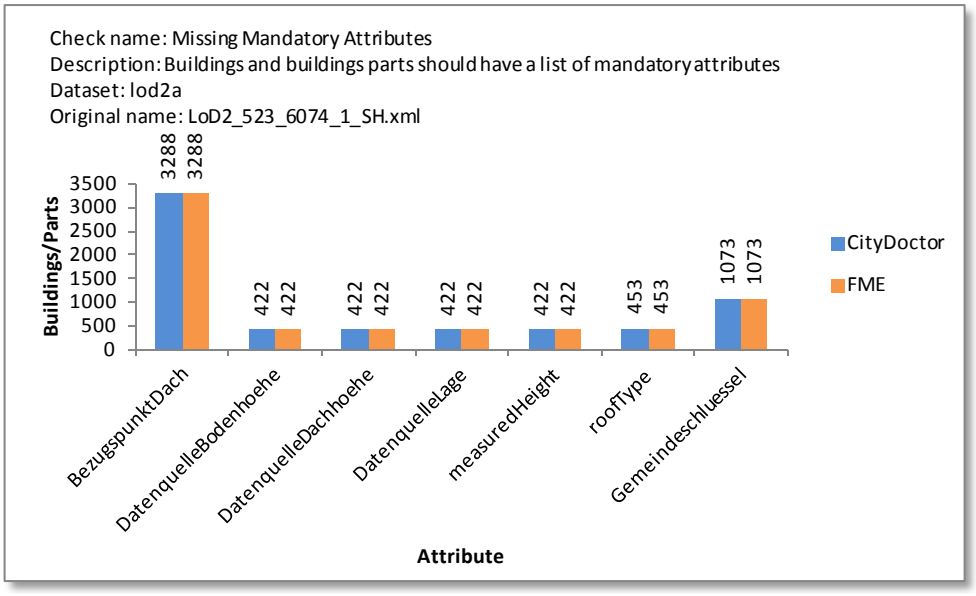
Google Sketchup offers a plug-in to inspect if 3D objects can form solids. The name of the plug-in is “TT Solid Inspector”. This tool reports only 15 entities with issues (Figure 3.27). Different reasons lead to the result. Sketchup automatically heals geometric and topological errors in entities. Moreover, entities are imported in as grouped geometry. They need to be exploded otherwise TT Solid Inspector cannot analyze the total of the entities. Sketchup flattens the building/building part relationship and creates one entity. TT Solid Inspector reports the common polygons between building parts since they prevent the object to be watertight.



(a) (b)

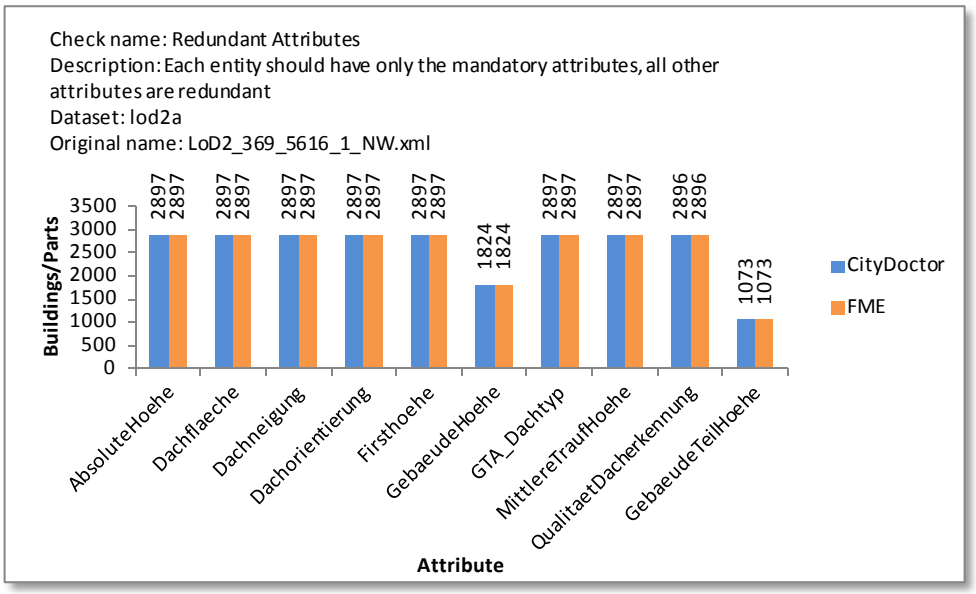
Figure 3.27: Use of Sketchup for further analysis. (a) TT Solid Inspector reports 15 entities with issues. (b) Building/BuildingPart relationship is flattened.

Both tools give the same results for the missing mandatory attributes (Graph 3.14). A large number of buildings/building parts is missing the “BezugspunktDach” attribute.



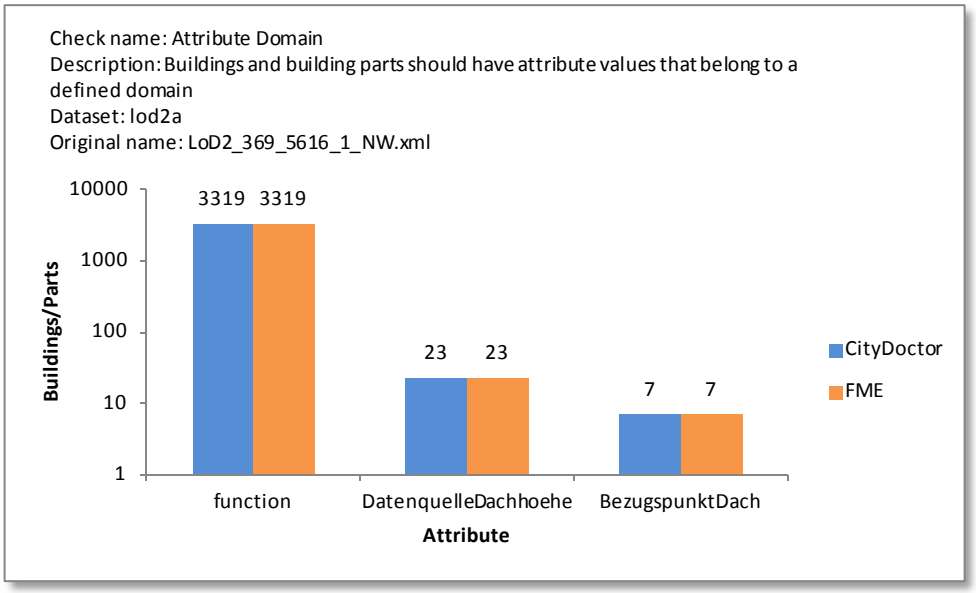
Graph 3.14: Missing attributes for lod2a dataset.

CityDoctor and FME report ten attributes as a redundant a large number of buildings/building parts (Graph 3.15).



Graph 3.15: Redundant attributes for lod2a dataset.

The attributes function, DatenquelleDachhoehe, and BezugspunktDach in lod2a dataset have values that do not belong in a domain defined by AdV. In Table 3.13 a snippet with the erroneous values can be seen.



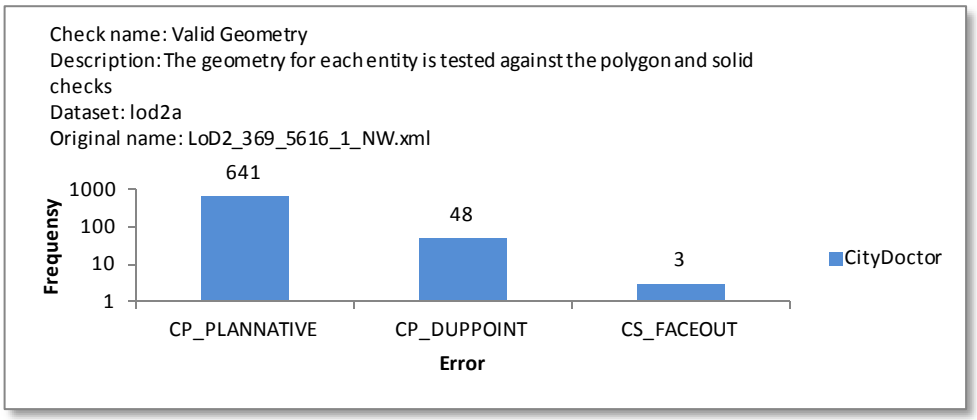
Graph 3.16: Attributes with values that do not belong to a defined domain.

GML ID	Attribute Name	Value
DENW_01a0c420-8818-4213-8610-7af0b21e2843	function	11_1003
DENW_01a7a7b2-93af-4a4a-8608-c47196650fb9	DatenquelleDachhoehe	9999
DENW_0b507e8e-99f1-48e2-bb4e-4cebd8e7f7dc	BezugspunktDach	9998

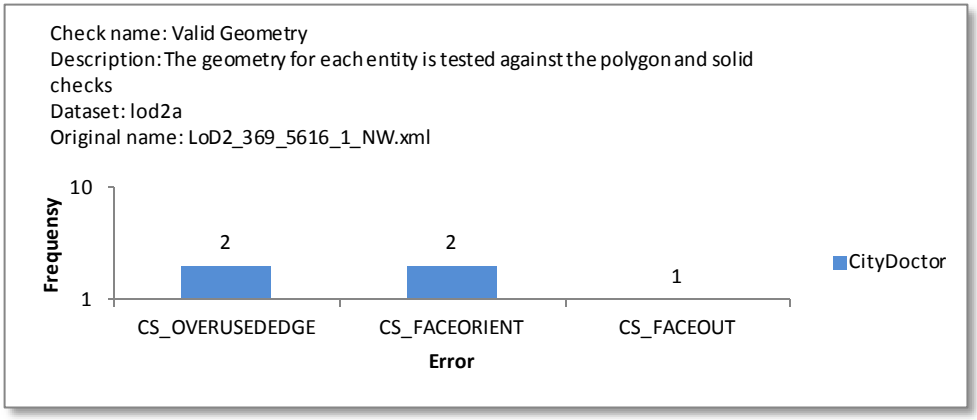
Table 3.13: Detail from raw results shows the erroneous values of attributes.

The valid geometry check reports errors in lod2a dataset. FME finds 53% more individual errors in total than CityDoctor (Graph 3.17).

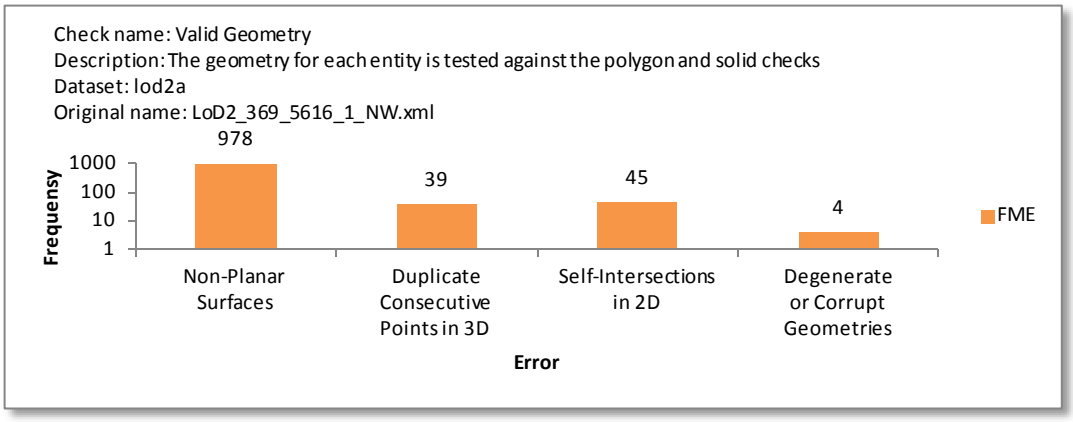
The error distribution shows that non-planar surfaces predominate. FME allows the user to define a tolerance value for the non-planar surfaces. In this case FME uses the same tolerance with CityDoctor, 0.01 meters. Even though the threshold that both tools use for the planarity is the same, the error frequency differs. FME reports 52.5% more planarity errors than CityDoctor.



(a)



(b)

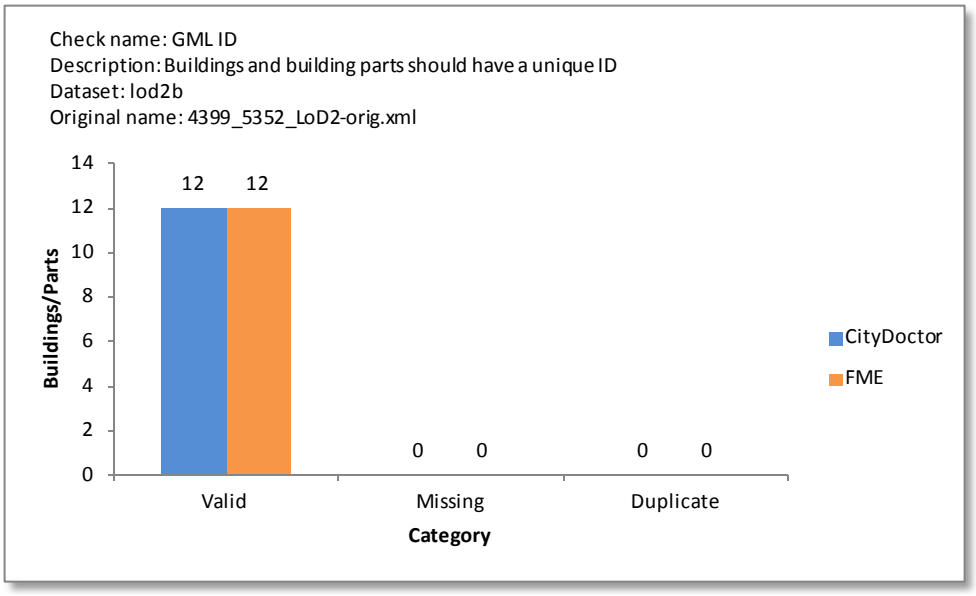


(c)

Graph 3.17: Geometry validation results for lod2a dataset. (a) CityDoctor first iteration results. (b) CityDoctor second iteration results. (c) FME results.

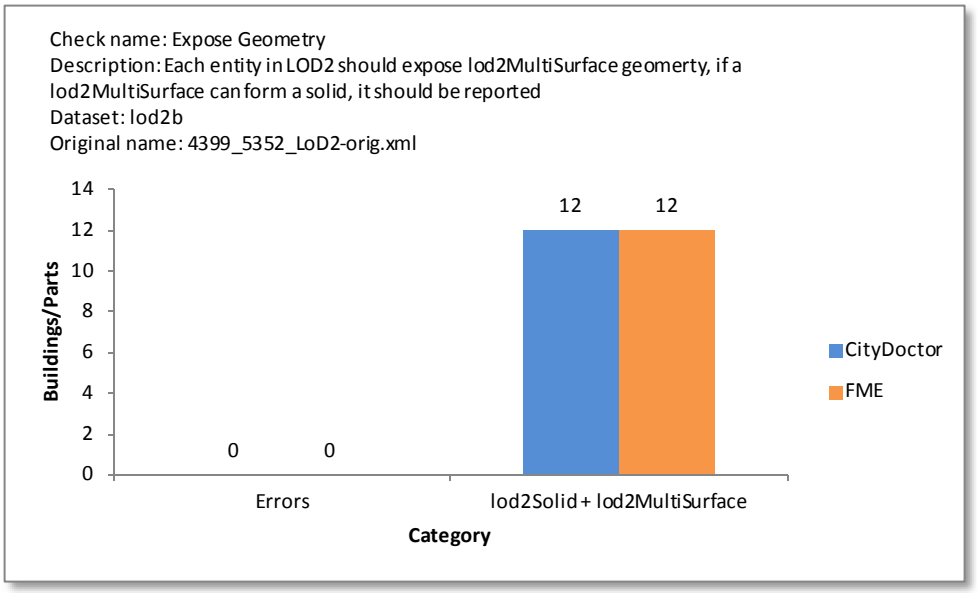
3.4.4 Results for lod2b dataset

In lod2b dataset CityDoctor and FME give the same results as appears in Graph 3.18. All buildings/building parts have unique IDs. None of them have missing or duplicate IDs.



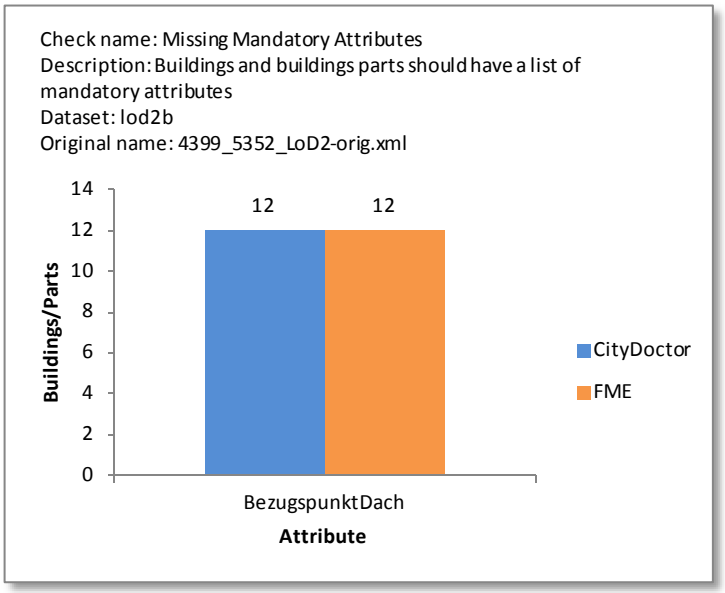
Graph 3.18: GML ID validation result for lod2b dataset.

All buildings/building parts expose the expected geometry type for the level of detail they belong. Any other geometry type is not present in this dataset (Graph 3.19).



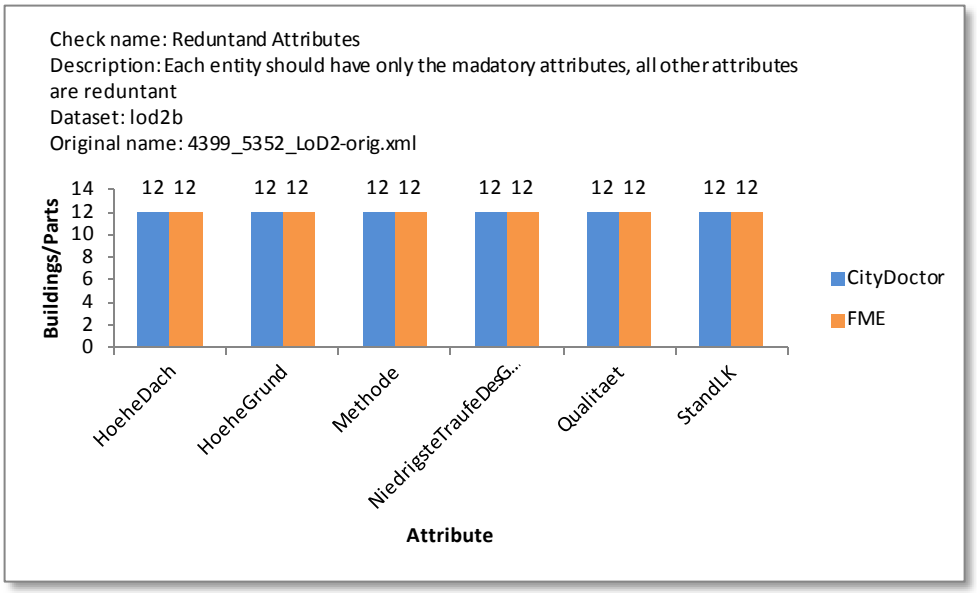
Graph 3.19: Expose Geometry validation result for lod2b dataset.

The results for the missing mandatory attributes are identical (Graph 3.20). 12 buildings/building parts report the BezugspunktDach attribute as missing.



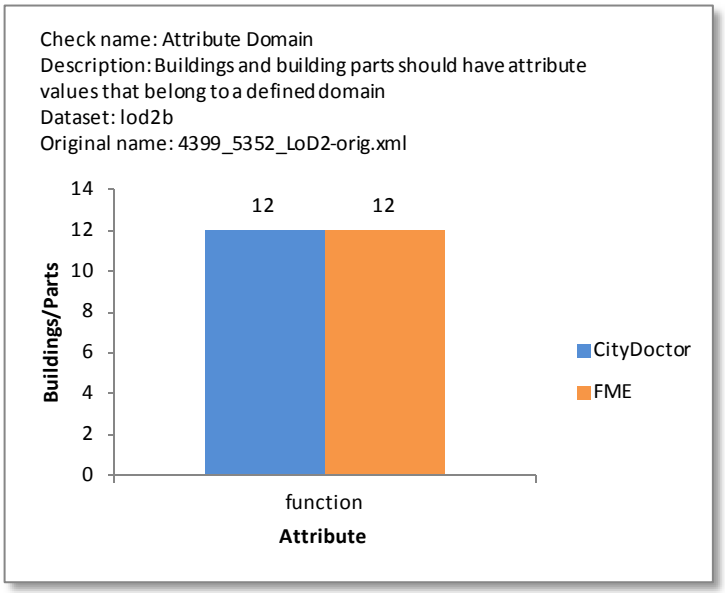
Graph 3.20: Missing attributes for lod2b dataset.

CityDoctor and FME report six attributes as a redundant in 12 buildings/building parts (Graph 3.21).



Graph 3.21: Redundant attributes for lod2b dataset.

The attribute function in lod2b dataset has values that do not belong in a domain defined by AdV (Graph 3.22). In Table 3.14 a snippet with the erroneous values can be seen.



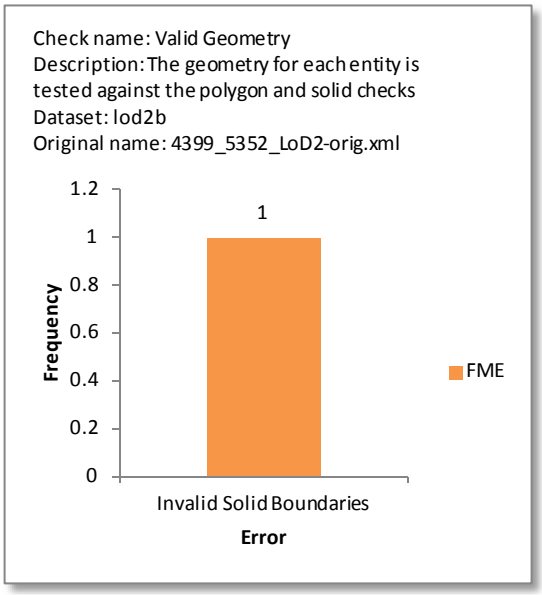
Graph 3.22: Attributes with values that do not belong to a defined domain.

GML ID	Attribute Name	Value
DEBY_LOD2_1925827	function	1001
DEBY_LOD2_1925828	function	1002

GML ID	Attribute Name	Value
DEBY_LOD2_1925829	function	1002

Table 3.14: Detail from raw results shows the erroneous values of attributes.

Only FME reports one error in valid geometry check in lod2b dataset (Graph 3.23). CityDoctor reports no errors. The erroneous building, with id: DEBY_LOD2_1925829, appears as solid in Sketchup and the Solid Inspector plug-in reports no error (Figure 3.28).



Graph 3.23: Geometry validation results for lod2b dataset.

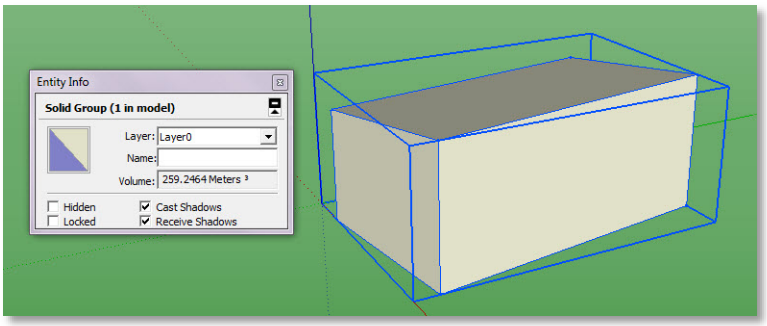
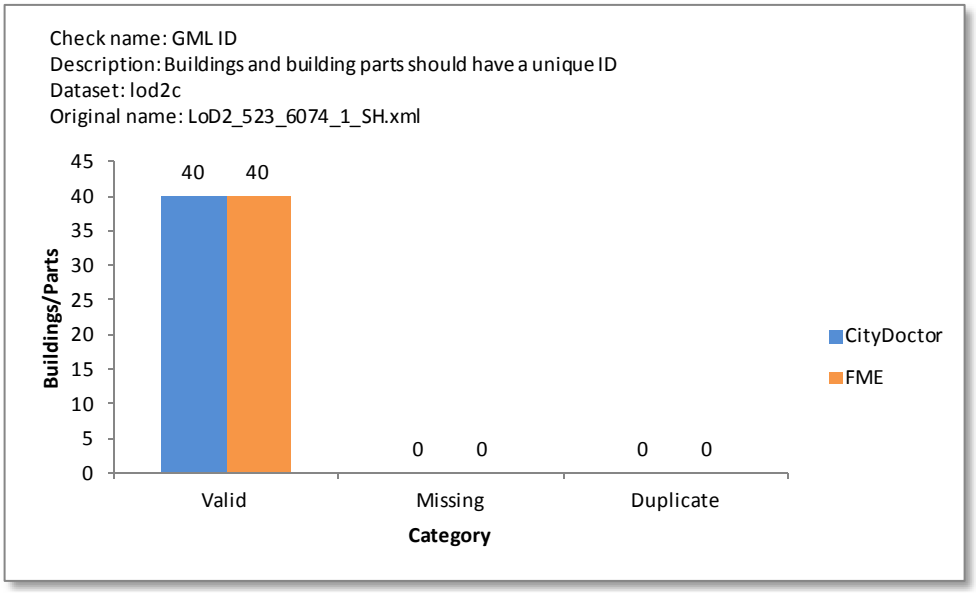


Figure 3.28: Sketchup is useful tool to inspect if an object can form solid.

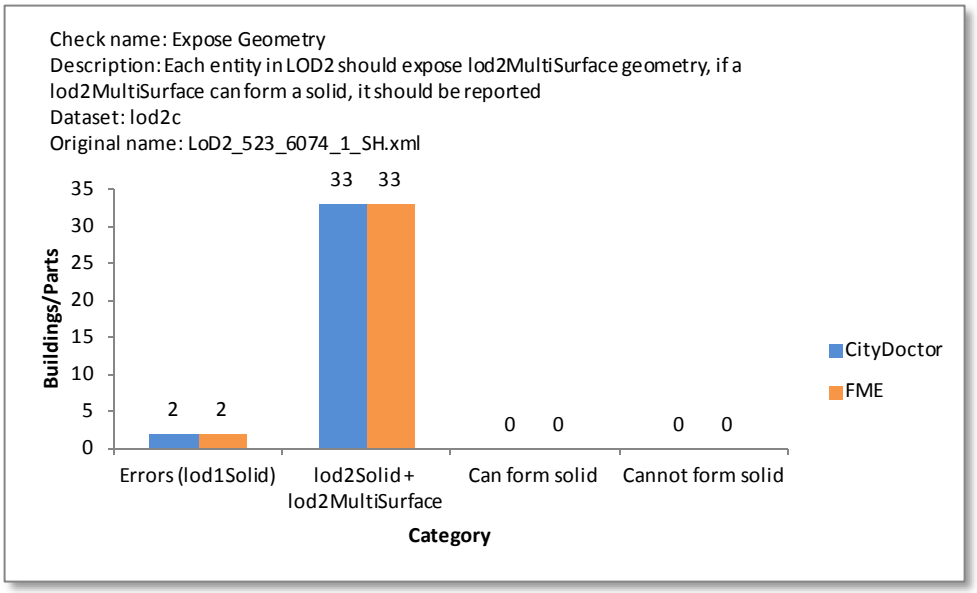
3.4.5 Results for lod2c dataset

In lod2c dataset CityDoctor and FME give the same results as appears in Graph 3.24. All buildings/building parts have unique IDs. None of them have missing or duplicate IDs.



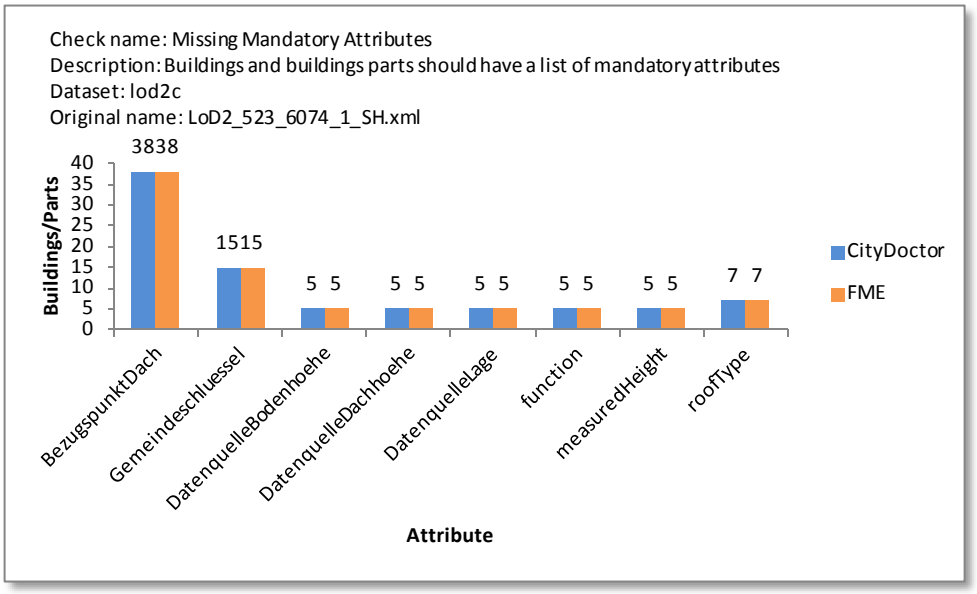
Graph 3.24: GML ID validation result for lod2c dataset.

Both tools report two buildings/building parts as erroneous because they expose lod1Solid geometry. The rest buildings/building parts expose the expected geometry type for the level of detail they belong. Any other geometry type is not present in this dataset (Graph 3.25).



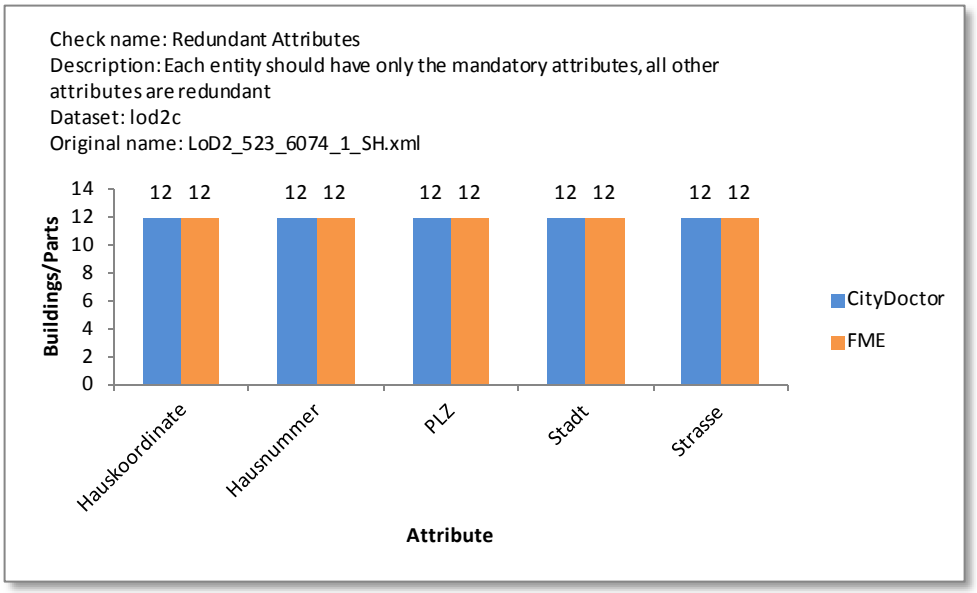
Graph 3.25: Expose Geometry validation result for lod2c dataset.

The results for the missing mandatory attributes are the same (Graph 3.26). Most of the buildings/building parts report the BezugspunktDach attribute as missing.



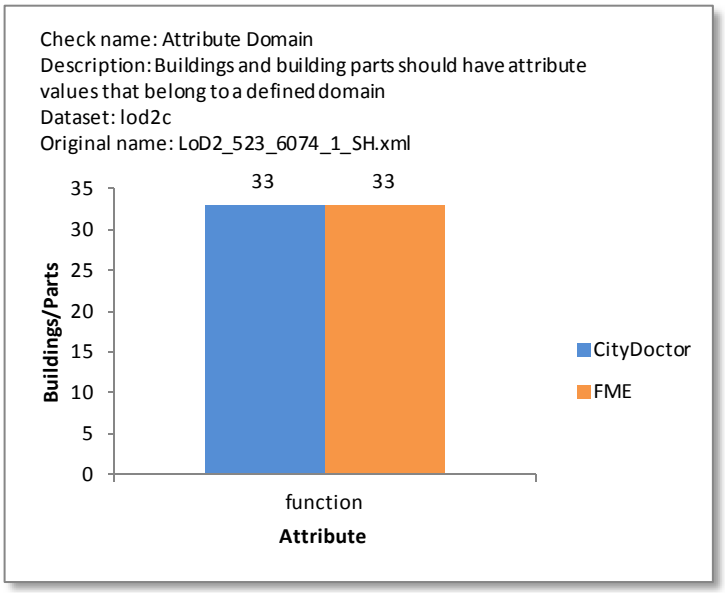
Graph 3.26: Missing attributes for lod2c dataset.

CityDoctor and FME report five attributes as a redundant in 12 buildings/building parts (Graph 3.27).



Graph 3.27: Redundant attributes for lod2c dataset.

The attribute function in lod2c dataset has values that do not belong in a domain defined by AdV (Graph 3.28). In Table 3.15, a snippet with the erroneous values can be seen. Erroneous values have a prefix code followed by an underscore and the actual value.

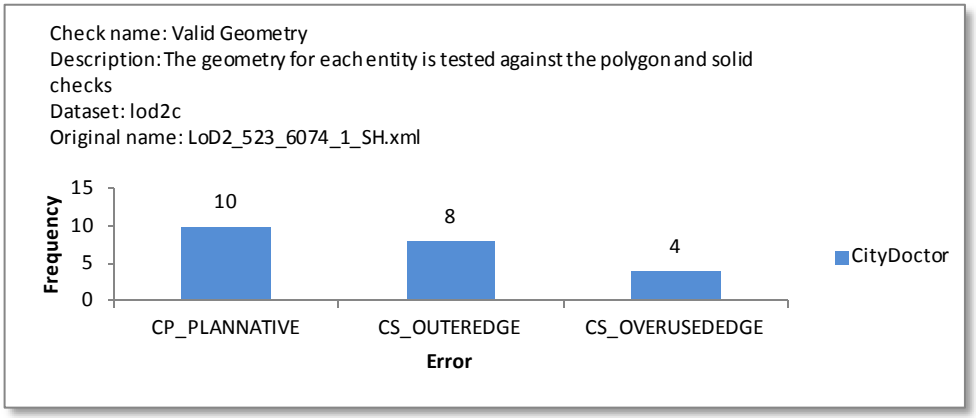


Graph 3.28: Attributes with values that do not belong to a defined domain.

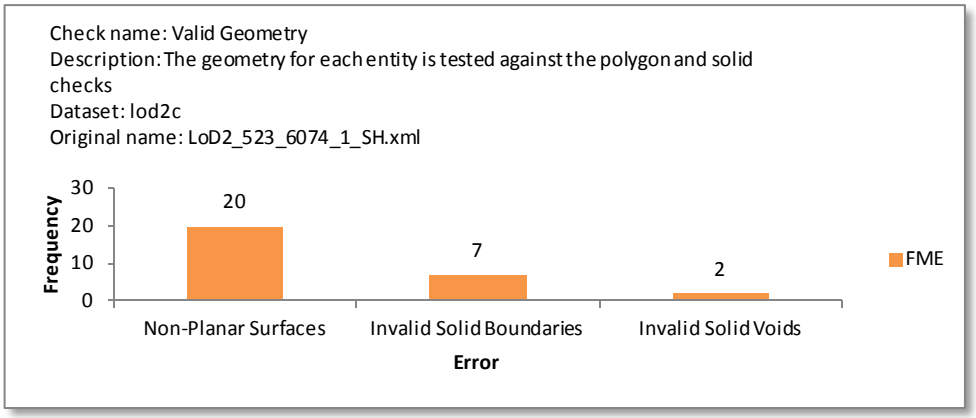
GML ID	Attribute Name	Value
DESH_00019730-0000-2000-20bb-0000013e4592	function	31001_2143
DESH_00019731-0000-2000-20bb-0000013e4592	function	31001_2143
DESH_00019732-0000-2000-20bb-0000013e4592	function	31001_1010

Table 3.15: Detail from raw results shows the erroneous values of attributes.

The valid geometry check reports errors in lod2c dataset. FME finds 31.8% more individual errors in total than CityDoctor (Graph 3.29). FME tends to report more planarity errors although the same threshold is used with CityDoctor, 0.01 meters. CityDoctor reports 12 solid errors: 8 OUTEREDGE and 4 OVERUSEDEDGE. FME reports 7 invalid solid boundaries and 2 invalid solid voids.



(a)



(b)

Graph 3.29: Geometry validation results for lod2c dataset. (a) CityDoctor first iteration results. (b) FME results.

The errors for the solid geometry can be correlated since an outer edge and an overused edge leads to an invalid B-rep solid. In detail, two buildings with IDs: DESH_85c36e92-a703-49b5-b8f4-2cde83a07d2f and DESH_9d1e3613-5819-49e9-ad87-399210dc9b8a, report in CityDoctor the outeredge and overusededge errors; the same buildings report 2 invalid solid voids and 2 invalid solid boundaries in FME (Figure 3.29).

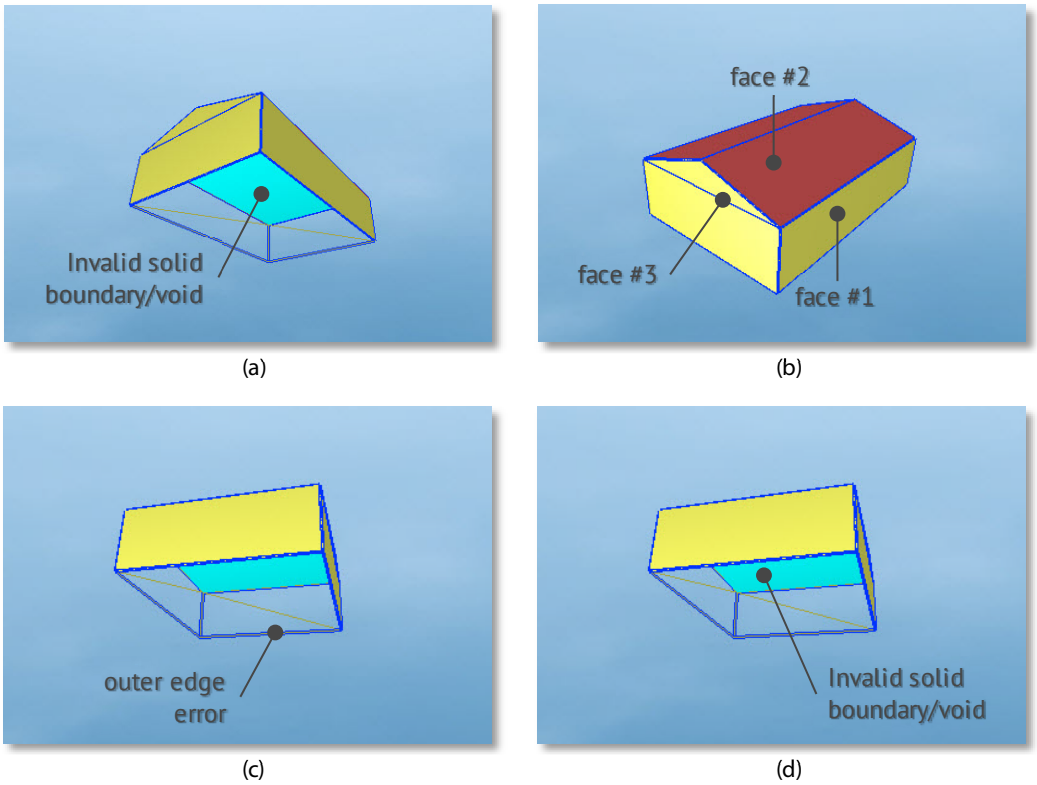


Figure 3.29: Erroneous buildings in lod2c dataset. (a) The first building cannot form solid. (b) Overused edge. (c) Outer edge error. (d) The second building cannot form solid.

The rest buildings that FME reports with invalid solid boundaries were extracted and imported in Google Sketchup to investigate if they can form solid. Sketchup represents those buildings as solid.

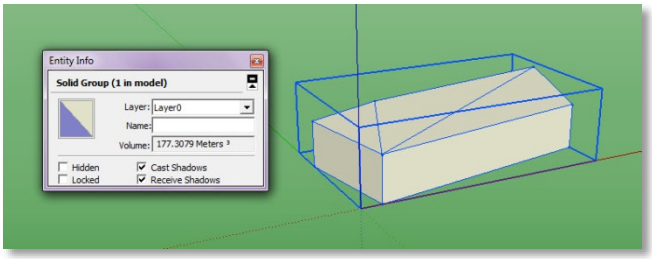
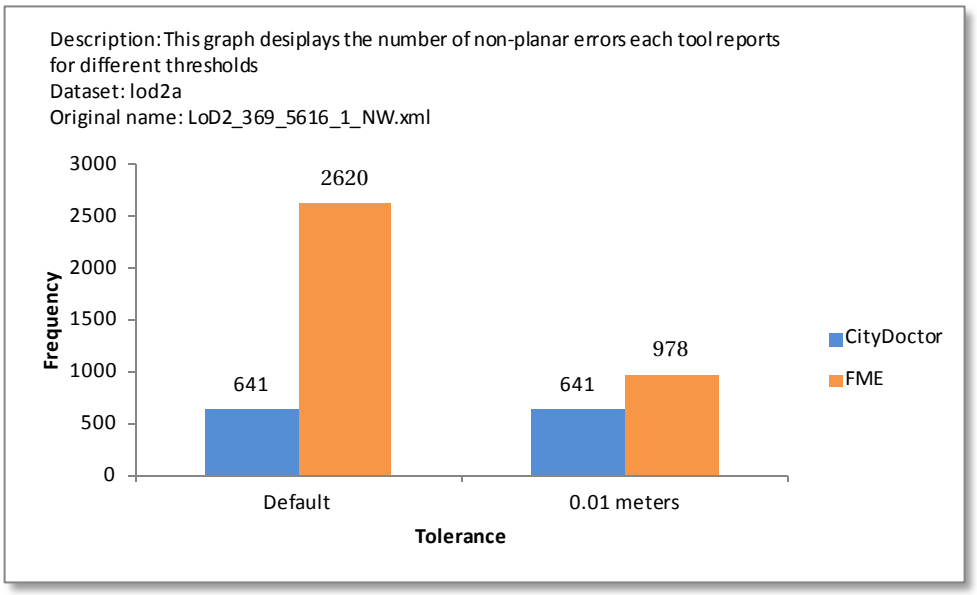


Figure 3.30: Sketchup reports an imported CityGML building as solid.

The overview of the results in the five point validation rule set shows similarities and differences. Naturally comes the question why such difference in the valid geometry results in lod2a dataset.

For the non-planar surfaces, FME allows the user to specify a fixed value for tolerance or an auto mode. CityDoctor, for the same check, has a default tolerance of 0.01 meters. Graph 3.30 shows that even with the same tolerance, FME reports more errors than CityDoctor. This result can be possibly explained by difference in the algorithm each tool uses for detecting non-planar faces.



Graph 3.30: Comparison of non-planar face check for different thresholds.

When FME detects a consecutive duplicate point in 3D, reports additionally a self-intersection in 2D. CityDoctor for the same error reports a duplicate point error only (Figure 3.31).

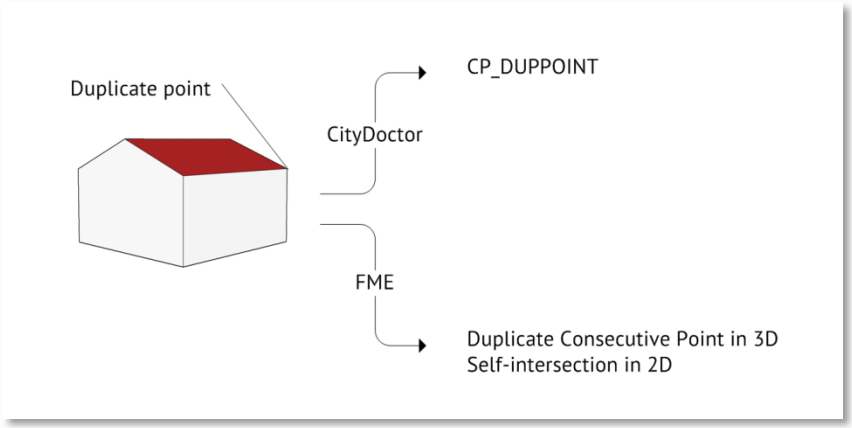


Figure 3.31: How a duplicate point is reported in both tools.

In cases like in Figure 3.32 FME reports a self-intersection in 2D whereas CityDoctor reports a duplicate point.

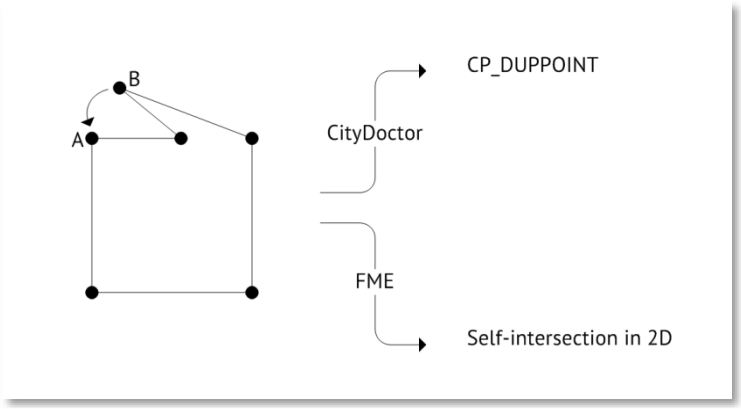


Figure 3.32: Point A coincides with point B. Both tools report a different error.

The geometry a building/building part exposes also influences the reported errors. For example, FME reports different errors for a building with a self-intersection that exposes multiple geometries (Figure 3.33).

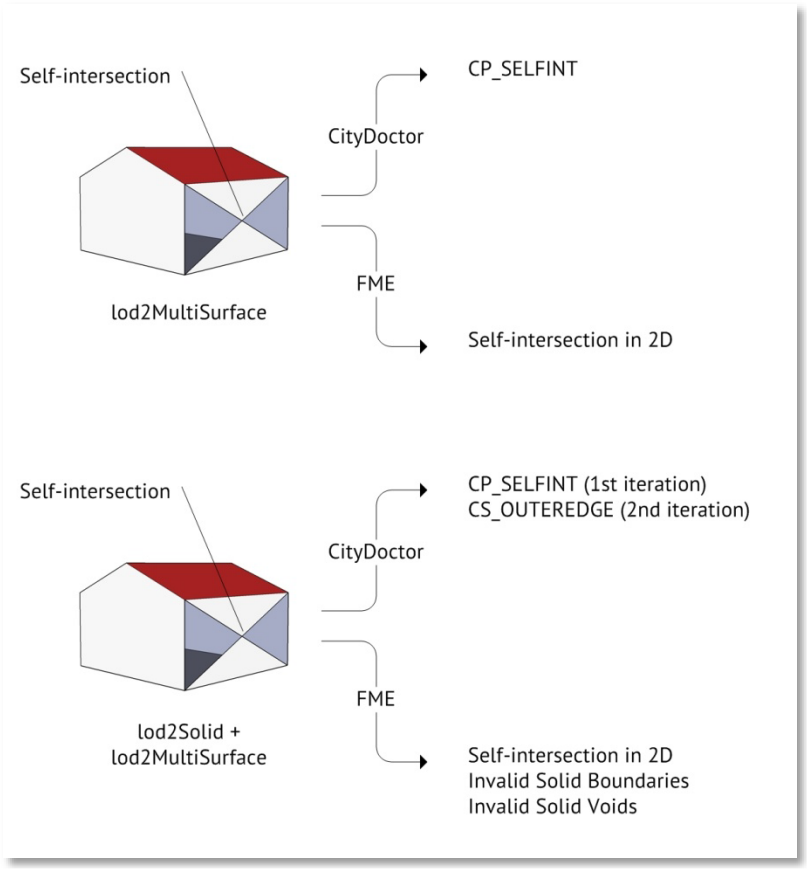


Figure 3.33: The exposed geometry influences the reported errors.

4 CityServer3D New Quality Approach

This chapter describes the methodology and the basic steps of integration of CityDoctor validation modules in CityServer3D.

4.1 CityServer3D

CityServer3D is a software application for managing spatial information, available as a desktop and server application. Developed by Fraunhofer Institute for Computer Graphics research in Darmstadt, Germany, CityServer is written in Java.

CityServer3D can manage 2D/3D geographic data, perform simulations and be used for city planning, real estate, land management etc.

The desktop version of the CityServer3D is the Admin Tool. It is a desktop application with many capabilities of importing and exporting various formats. The graphical user interface takes advantage of the Eclipse Rich Client Platform privileges i.e. it is customizable in different views, the perspectives (Figure 4.1).

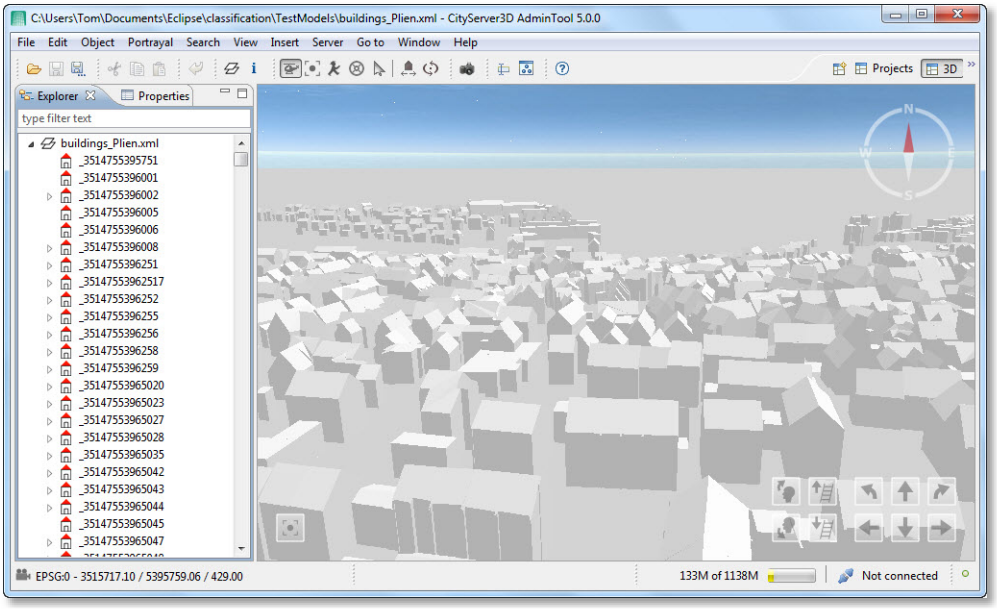


Figure 4.1: CityServer3D GUI.

The explorer tab displays the objects in the working dataset whereas the properties tab displays information about the selected object. The projects perspective allows the user to utilize the Graphical rule editor and define its own rules and actions performed to the current working set (Figure 4.2).

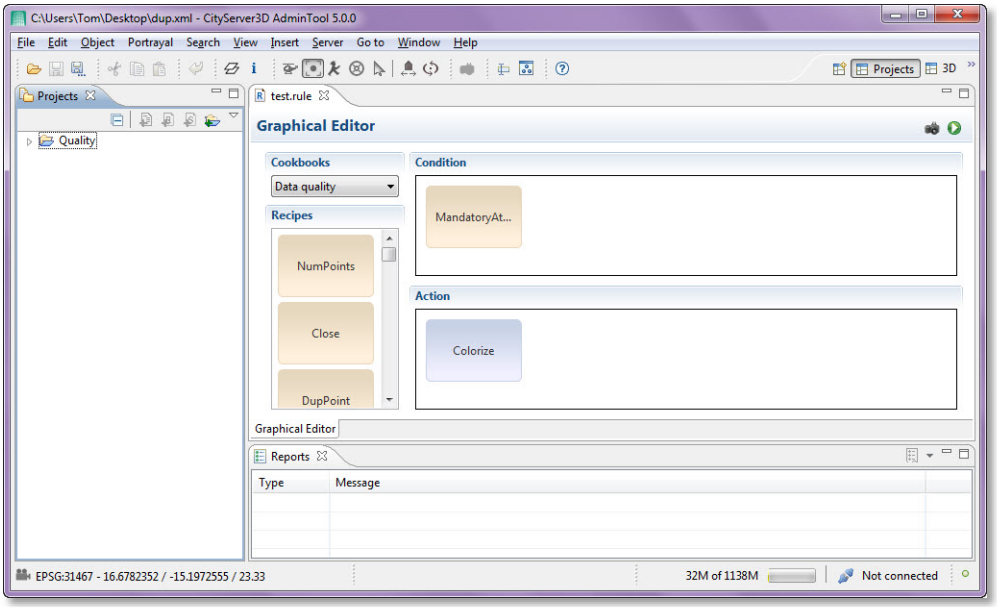


Figure 4.2: CityServer3D graphical rule editor.

The graphical editor displays a list of cookbooks and recipes. A cookbook contains several recipes. A recipe can be either a single action or a conditional

check to be performed to city object. Some recipes need to display additional graphical interface in order for the use r to adjust the recipe's settings (Figure 4.3). For example, the colorize recipe is going to colorize the entities that agree with the rule/rules the user defined.

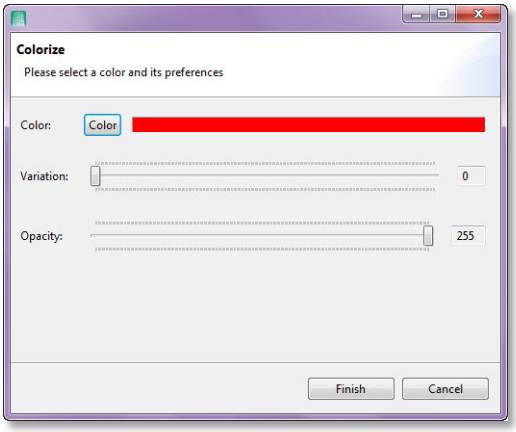


Figure 4.3: Colorize recipe's settings.

The check modules of CityDoctor are going to be the recipes of a cookbook named "Data Quality".

4.2 CityDoctor Implementation

An interface is used to bridge the buildings modeled in CityServer3D and CityDoctor. The CityDoctorConverter class implements the visitor architecture to convert a CityServer building/ building part into a CityDoctor building/ building part.

Each check module in CityDoctor becomes a bad quality recipe class in CityServer. Additionally for each recipe class a parameter class is created to reference the displayed text in the GUI (recipe name, description) and optionally the recipes parameters (Table 4.1). All the concrete quality recipes extend the BadQualityRecipe abstract class.

Recipe class	Recipe parameter class
BnbpifSolidRecipe.java	BnbpifSolidRecipeParam.java
CloseRecipe.java	CloseRecipeParam.java

Recipe class	Recipe parameter class
ConCompRecipe.java	ConCompRecipeParam.java
CoplanarSurfacesRecipe.java	CoplanarSurfacesRecipeParam.java
DupPointRecipe.java	DupPointRecipeParam.java
FaceOrientRecipe.java	FaceOrientRecipeParam.java
FaceOutRecipe.java	FaceOutRecipeParam.java
IsLOD1SolidRecipe.java	IsLOD1SolidRecipeParam.java
Lod1AsSolidRecipe.java	Lod1AsSolidRecipeParam.java
Lod1BuildParts1Recipe.java	Lod1BuildParts1RecipeParam.java
Lod1BuildParts2Recipe.java	Lod1BuildParts2RecipeParam.java
Lod1BuildParts3Recipe.java	Lod1BuildParts3RecipeParam.java
Lod1NumFloorsRecipe.java	Lod1NumFloorsRecipeParam.java
Lod2GroundRecipe.java	Lod2GroundRecipeParam.java
Lod2RoofRecipe.java	Lod2RoofRecipeParam.java
Lod2WallRecipe.java	Lod2WallRecipeParam.java
LowestEavesPointRecipe.java	LowestEavesPointRecipeParam.java
MandatoryAttrsRecipe.java	MandatoryAttrsRecipeParam.java
MsifSolidRecipe.java	MsifSolidRecipeParam.java
NullAreaRecipe.java	NullAreaRecipeParam.java
NumFacesRecipe.java	NumFacesRecipeParam.java
NumPointsRecipe.java	NumPointsRecipeParam.java
OuterEdgeRecipe.java	OuterEdgeRecipeParam.java
OverUsedEdgeRecipe.java	OverUsedEdgeRecipeParam.java
PlanarPatchRecipe.java	PlanarPatchRecipeParam.java
PlanDistAllRecipe.java	PlanDistAllRecipeParam.java
PlanDistRecipe.java	PlanDistRecipeParam.java
PlanNativeRecipe.java	PlanNativeRecipeParam.java
PlanTriRecipe.java	PlanTriRecipeParam.java
SelfIntNativeRecipe.java	SelfIntNativeRecipeParam.java
SelfIntRecipe.java	SelfIntRecipeParam.java
SurfaceAreaRecipe.java	SurfaceAreaRecipeParam.java
UmbrellaRecipe.java	UmbrellaRecipeParam.java

Table 4.1: Quality Recipes implemented in CityServer3D.

The source code uses GIT for version control. Local commits are pushed to Gerrit, a free web-based team software code review tool, where CityServer3D developers can assess and comment the uploaded code. If the pushed code is built successfully it is committed to the master source code project.

4.3 Results

After the implementation of the data quality cookbook, including all the bad quality recipes, the user is able to this cookbook in the graphical rule editor.

The efficiency of the new implementation was tested against CityGML datasets with known errors. In the example shown in Figure 4.4 a building with a semantic error (lod2Roof) was tested. The action colorizes the erroneous buildings into red (Figure 4.5).

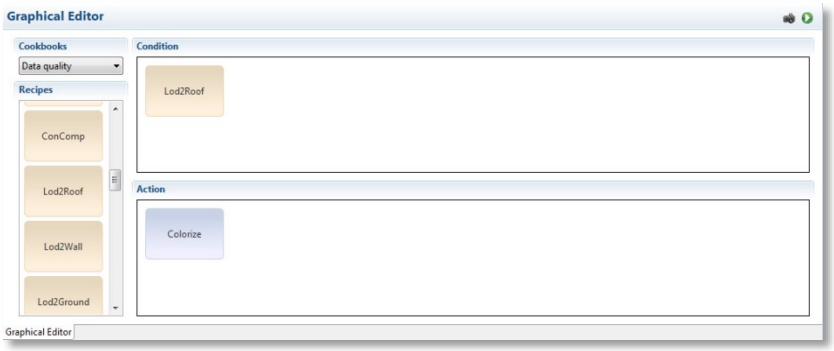


Figure 4.4: Condition and Action selection in the Graphical rule editor.

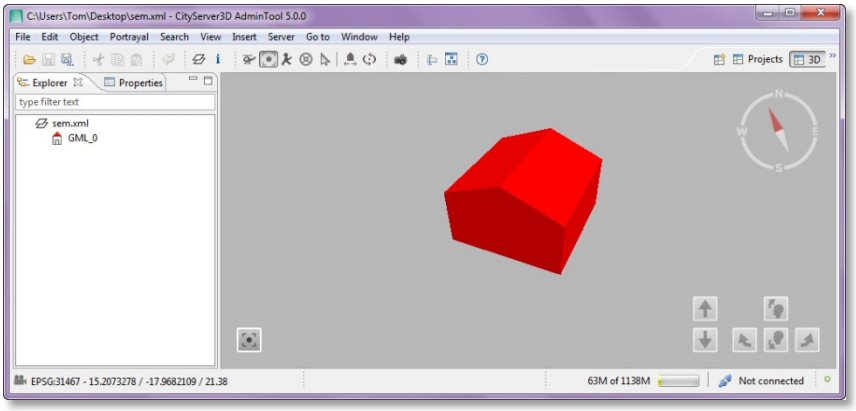


Figure 4.5: Erroneous building is colorized in red.

5 Conclusion

The scope of this thesis was to evaluate two validation tools for CityGML building entities and to integrate CityDoctor's validation modules into CityServer3D. All objectives of the thesis were fully achieved. A case study of AdV datasets with specific requirements and a 5 point rule set produced results in an evaluation framework.

In all validation points, except the valid geometry rule, CityDoctor and FME totally agree. Analyzing the valid geometry rule, it is difficult to say both tools either agree or disagree in their results. CityDoctor and FME handle an extremely polymorphic format in different ways. The inadequate error mapping is a second reason the results do not converge. Analyzing the geometry results from a logical perspective, it seems that both tools try to interpret an erroneous situation with different terms.

Moreover, the validation process can be characterized as software free only if FME is extended, using the Python scripting language, to support semantic error detection.

The challenges CityGML creates, need constant development of tools that handle its validation levels. CityDoctor can take advantage of becoming a rich client application and utilizing different perspectives for CityGML validation: Schema Validation perspective, geometry/semantic validation perspective, and attribute validation perspective. In that way, CityDoctor will be transformed from a validation tool to a validation suite, offering adjustable, rich content graphical user interface to match the needs of any user.

References

Blaauboer, J., Goos, J., Ledoux, H., Penninga, F., Reuvers, M., Stoter, J., et al. (2012). *Technical specifications for the reconstruction of 3D IMGeo CityGML data*.

Fawcett, J., Quin, L., Ayers, D., & Tegtmeyer, K. (2012). Beginning XML, 5th edition

Harold, E. R. (2001). XML bible, second edition

OGC. (2012). OGC City Geography Markup Language (CityGML) Encoding Standard. Open Geospatial Consortium (OGC) document 12-019.

Ostermann, A., & Wanner, G. (2012). Geometry by its history, Undergraduate texts in mathematics,pp. xii, 437 p.). Available from http://dx.doi.org/10.1007/978-3-642-29163-0

Raphael (Artist). (1509). *The School of Athens*.

safe.com. (2014a). FME Desktop Tutorial. Retrieved January, 10, 2014, from http://cdn.safe.com/training/tutorials/FME_Desktop_Tutorial_with_Data.zip

safe.com. (2014b). FME Transformer Reference Guide. Retrieved January, 10, 2014, from http://cdn.safe.com/resources/fme/FME-Transformer-Reference-Guide.pdf

Silva, V. (2009). *Practical Eclipse Rich Client Platform projects*. New York, N.Y.: Apress.

snowflakesoftware.com. (2014). Background Concepts - XML, GML and Application Schemas. Retrieved January 6, 2014, from http://wiki.snowflakesoftware.com/display/GL17DOC/Background+Concepts+-+XML,+GML+and+Application+Schemas

Vohra, A., Vohra, D., & Gowda, B. (2006). Pro XML development with Java technology, The expert's voice in Java technology

w3schools.com. (2014). XML Validation. Retrieved January 4, 2014, from http://www.w3schools.com/xml/xml_dtd.asp

Wyke, R. A., & Watt, A. (2002). *XML Schema Essentials*. New York: John Wiley.