



Ingenieur fakultät Bau Geo Umwelt  
Lehrstuhl für Geoinformatik

# **Domain Extendable 3D City Models – Management, Visualization, and Interaction**

Zhihang Yao

Vollständiger Abdruck der von der Ingenieur fakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Uwe Stilla

Prüfer der Dissertation:

1. Prof. Dr. Thomas H. Kolbe
2. Prof. Dr.-Ing. Liqiu Meng
3. Prof. Dr. Peter van Oosterom  
Delft University of Technology

Die Dissertation wurde am 01.02.2019 bei der Technischen Universität München eingereicht und durch die Ingenieur fakultät Bau Geo Umwelt am 20.12.2019 angenommen.



## Acknowledgements

During the past four years, this work has been carried out in large parts within the research project “*Fachlich erweiterbare 3D-Stadtmodelle – Management, Visualisierung und Interaktion*” funded by the company CADFEM GmbH, whom I would like to thank for its support firstly.

Next, I would like to express my sincere appreciation to Prof. Matthäus Schilcher who brought me into the field of Geographic Information System during my master studies and gave me the opportunity to start my research career at the Chair of Geoinformatics at Technical University of Munich (TUM). This thesis would not have been possible without his kind advice and encouragement in every phase of the dissertation work throughout the past four years.

My special thanks go to my PhD supervisor Prof. Thomas H. Kolbe, who gave me this challenging research topic and devote his time to consistently help me in training my soft skills to build my confidence towards the completion of this dissertation. His constructive criticism and valuable remarks from both scientific and engineering perspectives have tremendously helped me to establish the correct research direction and to dig deeper into many different interesting fields and tasks. I deeply appreciate his enlightening supervision which extended my scope of technical knowledge and which will benefit my long-term career development.

I would also like to give my sincere thanks to my two PhD co-supervisors Prof. Liqiu Meng from TUM and Prof. Peter van Oosterom from TU-Delft for carefully reviewing my thesis as well as for giving their valuable comments which guided me improving and finishing my thesis.

Besides my supervisors, I further want to thank my colleagues from the Chair of Geoinformatics for their great help, which let me to gain the insights into diverse research aspects. First, I owe my thanks to Dr. Tatjana Kutzner for the fruitful discussions on the technical implementation of CityGML extensions which is one of the key research aspects in my thesis. I am grateful to Dr. Andreas Donaubaue, who has been continuously paying attention to my research work and also provided valuable feedback after reviewing my thesis. I also thank Maximilian Sindram, Kanishk Chaturvedi, Son Nguyen, and my former colleagues Dr. Robert Kaden and Dr. Aftab Khan for their efforts in completing many joint articles.

The colleagues from the company virtualcitySYSTEMS GmbH (a subsidiary of the company CADFEM GmbH) have also offered me incredible support in completing this thesis. I am especially grateful to Dr. Claus Nagel for his tremendous contribution to the implementation of the CityGML Import/Export tool which is mentioned in many places within this thesis and plays an essential role in enhancing the functionalities of the developed framework. I also thank Jannes Bolling for the fruitful discussions on designing one of the kernel APIs of the developed 3D web client.

Last, I owe the deepest gratitude to my parents for their continual support and understanding, and all my love goes to my wife Xue Chen for her infinite patience, care, and love that accompanied me through the tough period.



## Contents

<b>Acknowledgements .....</b>	<b>3</b>
<b>Contents .....</b>	<b>5</b>
<b>Abbreviations .....</b>	<b>7</b>
<b>List of Figures .....</b>	<b>9</b>
<b>List of Tables .....</b>	<b>14</b>
<b>Summary .....</b>	<b>15</b>
<b>Zusammenfassung .....</b>	<b>17</b>
<b>Chapter 1 Introduction.....</b>	<b>19</b>
1.1 Motivation.....	19
1.2 Research Hypotheses and Questions .....	21
1.3 Organization of the Thesis .....	21
<b>Chapter 2 Theoretical Background .....</b>	<b>23</b>
2.1 Geographic Information System (GIS).....	23
2.2 Spatial Database System .....	25
2.3 Cloud Computing.....	28
2.4 Geospatial Data Modelling .....	31
2.5 3D Graphics Visualization.....	37
2.6 Digital Virtual Globes.....	41
<b>Chapter 3 Management of Semantic 3D City Models .....</b>	<b>45</b>
3.1 CityGML.....	46
3.1.1 Overview.....	46
3.1.2 Main Features of CityGML.....	47
3.2 Management of CityGML using SRDBMS.....	52
3.2.1 Standard Approach for Mapping of OO-Models onto Relational Models .....	53
3.2.2 Advanced Approach for optimizing Relational Database Models .....	59
3.3 3D City Database (3DCityDB) .....	63
3.3.1 Overview.....	63
3.3.2 3DCityDB Insights.....	64
<b>Chapter 4 Management of Domain Extendable 3D City Models .....</b>	<b>71</b>
4.1 Extending the CityGML Data Model .....	72
4.1.1 CityGML ADE Insights .....	73
4.1.2 Development of CityGML ADEs .....	74
4.1.3 Extending the 3DCityDB for CityGML ADEs .....	75
4.2 Automatic Derivation of Relational Database Schemas for ADEs .....	76
4.2.1 Survey of existing Transformation Solutions.....	77
4.2.2 Relevant Concepts of Graph Transformation System .....	81
4.2.3 Concept of using GTS to automatically derive ADE Database Schemas .....	87
4.3 Implementation and Evaluation .....	89

4.3.1	Design of a Graph Transformation Environment .....	90
4.3.2	Extending the 3DCityDB for Managing CityGML ADEs .....	98
4.3.3	Example Application: CityGML-TestADE.....	103
4.3.4	Practical Applications: EnergyADE and UtilityNetworkADE .....	108
<b>Chapter 5</b>	<b>Visualization and Exploration of Large Semantic 3D City Models .....</b>	<b>109</b>
5.1	Visualization of Semantic 3D City Models .....	110
5.1.1	Creation of 3D Visualization Models.....	111
5.1.2	Tiling of 3D Visualization Models.....	117
5.1.3	Streaming and Visualization of Tiled 3D Visualization Models .....	122
5.2	Exploration of Semantic 3D City Models.....	124
5.2.1	Interaction with 3D City Model Objects .....	125
5.2.2	Coupling of 3D Visualization Models with Thematic Information.....	127
5.2.3	Implementation of a 3D Web Client .....	130
5.3	Example Applications .....	135
5.3.1	3D City Model of New York City.....	136
5.3.2	3D City Model of Berlin .....	137
5.3.3	3D City Model of Vorarlberg.....	138
<b>Chapter 6</b>	<b>Utilization of Domain Extendable 3D City Models.....</b>	<b>139</b>
6.1	Conceptual Considerations .....	140
6.1.1	Problems of the traditional System Solutions .....	140
6.1.2	A new Multi-Level System Architecture .....	141
6.2	System Implementation .....	143
6.2.1	Implementation of the Information Backbone .....	143
6.2.2	Implementation of the Application Level .....	144
6.3	Example Applications .....	148
6.3.1	Use Case 1: Solar Potential Simulation for the district LBBD.....	148
6.3.2	Use Case 2: Energy Atlas Berlin.....	152
<b>Chapter 7</b>	<b>Discussion, Conclusions, and Outlook .....</b>	<b>155</b>
7.1	Discussion .....	155
7.2	Contribution of the Thesis .....	159
7.3	Outlook and Future Research.....	161
	<b>Bibliography.....</b>	<b>163</b>
	<b>Appendix 1: Layered Graph Transformation Rules.....</b>	<b>175</b>
	<b>Appendix 2: XML Schema Definition File of the TestADE .....</b>	<b>191</b>
	<b>Appendix 3: SQL Definition of the TestADE Oracle DB-Schema.....</b>	<b>195</b>
	<b>Appendix 4: SQL Definition of the TestADE PostGIS DB-Schema .....</b>	<b>201</b>
	<b>Appendix 5: XML Definition of the TestADE Schema Mapping .....</b>	<b>209</b>

## Abbreviations

3DCityDB	3D City Database
AGG	Attributed Graph Grammar
ADE	CityGML Application Domain Extension
API	Application Programming Interface
B-Rep	Boundary Representation
BIM	Building Information Model
B3DM	Batched 3D Model
CZML	Cesium Markup Language
CPU	Central Processing Unit
CAD	Computer-aided Design
CRUD	Create, Read, Update, and Delete
CityGML	City Geography Markup Language
COLLADA	Collaborative Design Activity
CRS	Coordinate Reference System
DPO	Double-Pushout
DCC	Digital Content Creation
DTM	Digital Terrain Model
DSM	Digital Service Model
DBS	Database System
DBMS	Database Management System
ETL	Extract, Transform, and Load
EA	Enterprise Architect
ESRI	Environmental Systems Research Institute
FME	Feature Manipulation Engine
GPU	Graphics Processing Unit
GIS	Geographic Information System
GUI	Graphical User Interface
GFM	General Feature Model
GML	Geography Markup Language
GFM	General Feature Model
GTS	Graph Transformation System
glTF	GL Transmission Format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
INSPIRE	Infrastructure for Spatial Information in Europe
ISO	International Organisation for Standardisation

---

IaaS	Infrastructure as a Service
KML	Keyhole Markup Language
LOD	Level of Detail
LRU	Least Recently Used
LHS	Left-hand Side
MDA	Model-driven Architecture
NAC	Negative Application Condition
OGC	Open Geospatial Consortium
OMG	Object Management Group
OO	Object-oriented
PAC	Positive Application Condition
PIM	Platform Independent Model
PSM	Platform Specific Model
PaaS	Platform as a Service
RHS	Right-hand Side
REST	Representational State Transfer
SRDBMS	Spatially-enhanced Relational Database Management System
SQL	Structured Query Language
SaaS	Software as a Service
SDBS	Spatially-enhanced Database System
SPO	Single-Pushout
TMS	Tile Map Service
TC211	ISO Technical Committee 211
TUM	Technische Universität München
UML	Unified Modeling Language
URI	Unique Resource Identifier
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
WFS	OGC Web Feature Service
WMS	OGC Web Map Service
WPS	OGC Web Processing Service
WGS 84	World Geodetic System 1984
WebGL	Web Graphics Library
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition
X3D	Extensible 3D Graphics
XPath	XML Path Language



## List of Figures

Figure 1: Relevant components of a typical GIS environment (cf. Jung 2008).....	24
Figure 2: Application structure based on a spatial database system (cf. Brinkhoff 2005).....	25
Figure 3: An example of R-Tree spatial index (cf. Guttman 1984) .....	27
Figure 4: Conceptual system architecture of Cloud Computing (cf. Buyya et al. 2008) .....	29
Figure 5: An example of the three types of Cloud-based services using the Google Cloud ecosystem .....	30
Figure 6: Conceptual idea of the interoperable data dissemination using a common data model in GIS .....	32
Figure 7: Relationship between PIM and PSMs in a Model-Driven Architecture.....	33
Figure 8: Spatio-semantic representation of a building based on ISO 19107 and 19109 standards.....	34
Figure 9: Relationship between the conceptual models from the ISO 19100 series and their GML encodings (ISO 19136:2007).....	35
Figure 10: 3D Graphics Rendering Pipeline (Chuan 2012) .....	38
Figure 11: API architecture of the CesiumJS framework library (Cozzi 2015).....	42
Figure 12: Graphical user interface of the Cesium Viewer.....	43
Figure 13: Overview of the CityGML modules (cf. Gröger et al. 2012) .....	46
Figure 14: The LOD concept defined by CityGML (Biljecki et al. 2016).....	47
Figure 15: Example of realizing an aggregation hierarchy using CityGML's grouping concept .....	48
Figure 16: Graphical UML notation of the CityGML geometry model (Gröger et al. 2012)..	49
Figure 17: Coherence of semantics and geometry in CityGML (Stadler & Kolbe 2007) .....	50
Figure 18: Example of using CityGML External References for linking with remote data repositories .....	51
Figure 19: Relationship between CityGML and ADE modules .....	52
Figure 20: Mapping a Class onto a table.....	54
Figure 21: Mapping each class of an inheritance hierarchy onto a separate database table ...	55
Figure 22: Mapping two classes with inheritance relationship onto a single database table ...	56
Figure 23: Mapping N:0..1 relationship between two different classes (variant 1).....	56
Figure 24: Mapping N:0..1 relationship between two classes with a shared table (variant 2).	56
Figure 25: Mapping N:0..1 relationship of the same class (variant 3).....	57
Figure 26: Mapping 1:N relationship between two different classes (variant 1).....	57
Figure 27: Mapping 1:N relationship between two classes with a shared table (variant 2).....	57
Figure 28: Mapping 1:N relationship between the same class (variant 3).....	58
Figure 29: Mapping M:N association relationship (Variant 1) .....	58
Figure 30: Mapping M:N association relationship (variant 2).....	58
Figure 31: Example of mapping an inheritance hierarchy onto one table .....	59

Figure 32: Example of mapping multiple classes onto one table.....	60
Figure 33: General idea for the mapping of an object-oriented model with the composite pattern onto an efficient relational database model.....	61
Figure 34: GML geometry types being used in the CityGML standard (cf. Kolbe et al. 2016) .....	62
Figure 35: Relevant components of the 3DCityDB Software Suite.....	64
Figure 36: Overview of the 3DCityDB database procedure packages.....	65
Figure 37: Software structure of the CityGML Import/Export Tool (cf. Stadler et al. 2009)..	66
Figure 38: Implementation of the 3DCityDB Web Feature Service .....	67
Figure 39: Workflow of generating KML/COLLADA/glTF visualization models.....	68
Figure 40: Workflow of generating spreadsheet from 3DCityDB.....	69
Figure 41: Workflow of using 3DCityDB web client coupled with Cloud-based online spreadsheets.....	70
Figure 42: Relationships between the ISO 19100 standard family and CityGML ADEs.....	73
Figure 43: Conceptual workflow for creating a dynamically extendable 3D geo-database for CityGML .....	76
Figure 44: General approach for deriving relational database schema based on model transformation .....	77
Figure 45: Conceptual workflow of using graph transformation system to perform a computation process.....	82
Figure 46: Conceptual diagram of the SPO-based approach (cf. Geiß et al. 2006).....	82
Figure 47: Example of the “dangling” edges resulted from a simple graph rewriting process	83
Figure 48: Conceptual diagram of the DPO-based approach (cf. Habel et al. 2001).....	84
Figure 49: Example of DPO approach for graph transformation.....	85
Figure 50: Conceptual diagram of graph transformation with type graph (cf. Taentzer et al. 2006).....	86
Figure 51: Conceptual workflow of layer-based graph transformation .....	87
Figure 52: General idea of deriving a relational database schema from a GML application schema by means of graph transformation.....	88
Figure 53: Software structure of the developed graph-based converter tool for generating relational database schema and schema mapping file from a given ADE application schema	89
Figure 54: Meta-model of the GML application schema according to the ISO 19136.....	90
Figure 55: Meta-model of the relational database model.....	92
Figure 56: An excerpt of the meta-graph for representing the model mapping structure .....	93
Figure 57: Rule 1: Mapping ADE class onto table .....	94
Figure 58: Rule 2: Mapping two ADE classes with inheritance relationship onto one table ..	95
Figure 59: Negative application conditions of the Rule2.....	95
Figure 60: Rule 3: Mapping Inheritance to a foreign key constraint .....	95

Figure 61: Rule 4: mapping two classes which have a composition relationship and are mapped onto the one table.....	96
Figure 62: Rule 5: Mapping simple attribute property onto a table column .....	96
Figure 63: Rule 6: Initializing a node for representing the 3DCityDB table "SURFACE_GEOMETRY" .....	97
Figure 64: Rule 7: Mapping B-Rep-based geometry property onto a foreign key column referencing to the SURFACE_GEOMETRY table.....	97
Figure 65: New conceptual 3DCityDB database structure for handling CityGML ADEs .....	98
Figure 66: Technical implementation of the 3DCityDB Metadata Module in a relational diagram.....	99
Figure 67: Software structure of the extended CityGML Import/Export Tool .....	101
Figure 68: Workflow of deregistration for a CityGML ADE from a database instance.....	103
Figure 69: An artificial CityGML ADE for testing the developed graph-based transformation approach .....	105
Figure 70: Concept of grouping and merging GML, CityGML, and ADE classes for deriving compact relational database models. Note that the classes of the groups 1, 8, 9, 13, and 15 are standard GML and CityGML classes, which have already been mapped to the default 3DCityDB tables. ....	106
Figure 71: Relational diagram of the automatically derived relational database of the artificial ADE using the developed graph-based converter tool.....	107
Figure 72: Report of Database instance including EnergyADE and UtilityNetworkADE database schemas.....	108
Figure 73: Conceptual structure for the Web-based geo-visualization of 3D city models ....	110
Figure 74: Workflow of generating 3D visualization models.....	111
Figure 75: Comparison between the coordinates of different reference system types. Note that the latitude 'La' of the point 'P' is geodetic latitude which is the angle between the corresponding surface normal and the equatorial plane, rather than the angle (called 'geocentric loatitude') between the equatorial plane and the line connecting the point 'P' and the orgin 'O'.....	112
Figure 76: Workflow of interacting with Elevation API.....	113
Figure 77: Example of different display forms of the created 3D visualization models .....	115
Figure 78: Comparison of the different visual effects of the same 3D model with (the left figure) and without (the right figure) surface normal .....	115
Figure 79: Example of the first packing algorithm .....	116
Figure 80: Comparison of the generated texture atlas images created using the Basic (left) and TPIM (right) packing algorithms .....	117
Figure 81: Excerpt of the TMS layout based on the Web Mercator Auxiliary Sphere.....	118
Figure 82: Typical spatial data structure supported by the 3D-Tiles standard (cf. Cozzi et al. 2019).....	119
Figure 83: Hierarchical directory structure for export of 2x3 tiles using the grid-based tiling layout.....	120

Figure 84: Strategy for determining the candidate data tiles that should be loaded according to the camera perspective projected onto the screen space .....	122
Figure 85: Efficient determination of which data tiles should be loaded according to the user-defined visibility range in screen pixel .....	123
Figure 86: Workflow of performing the determination and caching of the data tiles for efficient 3D visualization .....	124
Figure 87: Exploration of a building in a mash-up view using ‘Dual Maps’ (www.dualmaps.com) .....	125
Figure 88: Modelling of the hierarchical structure of 3D city models using the glTF-based B3DM format (cf. Schilling et al. 2016) .....	126
Figure 89: Example system architecture of coupling WFS with 3D visualization models in a 3D web map application.....	127
Figure 90: Coupling an online spreadsheet with a 3D visualization model via GMLID.....	128
Figure 91: Example system architecture of coupling online spreadsheet with 3D visualization models in web applications .....	129
Figure 92: Conceptual API for coupling multiple online spreadsheets with a 3D visualization model.....	130
Figure 93: System architecture of the developed 3D web client.....	131
Figure 94: User interface and the relevant GUI components of the 3D web client .....	132
Figure 95: Table manager for dynamically adding multiple online spreadsheet to a data layer .....	133
Figure 96: Idea of using a configuration spreadsheet for storing and loading linked 3D visualization models and online spreadsheets .....	134
Figure 97: Collaborative work using the 3D web client based on Cloud-based online spreadsheets (cf. Herreruella et al. 2012) .....	135
Figure 98: Example demo of visualizing 3D city model of New York City on the 3D web client .....	136
Figure 99: Example of visualizing 3D city model of Berlin on the 3D web client.....	137
Figure 100: Example of visualizing 3D landscape model of Vorarlberg on the 3D web client .....	138
Figure 101: Traditional Tree-tier architecture of a 3D Web GIS application (cf. Westra 2010) .....	140
Figure 102: Three-tier system architecture (cf. Yao et al. 2014) .....	141
Figure 103: Workflow of setting up the information backbone and urban analytics toolkit .	143
Figure 104: Sequence diagram of the process flow when performing a query on an online spreadsheet .....	145
Figure 105: Conceptual idea of using online spreadsheet to perform a simple spatial query	146
Figure 106: Example of creating a simple calculation engine using an online spreadsheet ..	146
Figure 107: Example of using two separate spreadsheets to perform calculations and simulations with secured data .....	147
Figure 108: User interface after loading the LBBDD 3D building models into the 3D web client .....	149

---

Figure 109: User-dialog for signing up using a valid Google Account .....	149
Figure 110: Example of performing a spatial query using a user-defined bounding box.....	150
Figure 111: Example of performing a simple query based on an attribute .....	150
Figure 112: Example of performing a complex query based on multiple attributes.....	151
Figure 113: Example of performing aggregation calculation on multiple numeric attributes	151
Figure 114: Example of the automatically generated report showing an overview of the statistic information.....	152
Figure 115: Example of showing the estimated energy demand for a selected building.....	153
Figure 116: Ad-hoc estimation of the heating energy demand for one building (cf. Yao et al. 2014).....	153

## List of Tables

Table 1: Comparison of the key features between different 3D visualization models .....	40
Table 2: Quantitative comparison of the number of tables generated from different software tools .....	104

## Summary

Domain-extendable semantic 3D city models are complex mappings and inventories of the urban environment which can be utilized as an integrative information backbone to facilitate a range of application fields like urban planning, environmental simulations, disaster management, and energy assessment. Today, more and more countries and cities worldwide are creating their own 3D city models based on the CityGML specification which is an international standard issued by the Open Geospatial Consortium (OGC) to provide an open data model and XML-based format for describing the relevant urban objects with regards to their 3D geometry, topology, semantics, and appearance. It especially provides a flexible and systematic extension mechanism called “Application Domain Extension (ADE)” which allows third parties to dynamically extend the existing CityGML definitions with additional information models from different application domains for representing the extended or newly introduced geographic object types within a common framework. However, due to the consequent large size and high model complexity, the practical utilization of country-wide CityGML datasets has posed a tremendous challenge regarding the setup of an extensive application system to support the efficient data storage, analysis, management, interaction, and visualization. These requirements have been partly solved by the existing free 3D geodatabase solution called ‘3D City Database (3DCityDB)’ which offers a rich set of functionalities for dealing with standard CityGML data models, but lacked the support for CityGML ADEs.

The key motivation of this thesis is to develop a reliable approach for extending the existing database solution to support the efficient management, visualization, and interaction of large geospatial data elements of arbitrary CityGML ADEs. Emphasis is first placed on answering the question of how to dynamically extend the relational database schema by parsing and interpreting the XML schema files of the ADE and dynamically create new database tables accordingly. Based on a comprehensive survey of the related work, a new graph-based framework has been proposed which uses typed and attributed graphs for semantically representing the object-oriented data models of CityGML ADEs and utilizes graph transformation systems to automatically generate compact table structures extending the 3DCityDB. The transformation process is performed by applying a series of fine-grained graph transformation rules which allow users to declaratively describe the complex mapping rules including the optimization concepts that are employed in the development of the 3DCityDB database schema.

The second major contribution of this thesis is the development of a new multi-level system which can serve as a complete and integrative platform for facilitating the various analysis, simulation, and modification operations on the complex-structured 3D city models based on CityGML and 3DCityDB. It introduces an additional application level based on a so-called ‘app-concept’ that allows for constructing a light-weight web application to reach a good balance between the high data model complexity and the specific application requirements of the end users. Each application can be easily built on top of a developed 3D web client whose functionalities go beyond the efficient 3D geo-visualization and interactive exploration, and also allows for performing collaborative modifications and analysis of 3D city models by taking advantage of the Cloud Computing technology. This multi-level system along with the extended 3DCityDB have been successfully utilized and evaluated by many practical projects.





## Zusammenfassung

Fachlich erweiterbare semantische 3D-Stadtmodelle sind komplexe Abbildungen und Datenbestände der städtischen Umgebung, die als ein integratives Informationsrückgrat genutzt werden können, um eine Reihe von Anwendungsfeldern wie z. B. Stadtplanung, Umweltsimulationen, Katastrophenmanagement und Energiebewertung zu ermöglichen. Heute schaffen immer mehr Länder und Städte weltweit ihre eigenen 3D-Stadtmodelle auf Basis des internationalen Standards CityGML des Open Geospatial Consortium (OGC), um ein offenes Datenmodell und ein XML-basiertes Format zur Beschreibung der relevanten Stadtobjekte in Bezug auf ihre 3D-Geometrien, Topologien, Semantik und Erscheinungen zur Verfügung zu stellen. Es bietet insbesondere einen flexiblen und systematischen Erweiterungsmechanismus namens „Application Domain Extension“ (ADE), der es Dritten ermöglicht, die bestehenden CityGML-Definitionen mit zusätzlichen Informationsmodellen aus verschiedenen Anwendungsdomänen dynamisch zu erweitern, um die erweiterten oder neu eingeführten Stadtobjekt-Typen innerhalb eines gemeinsamen Framework zu repräsentieren. Aufgrund der konsequent großen Datenmenge und hohen Modellkomplexität bei der praktischen Nutzung der landesweiten CityGML-Datensätze wurden jedoch enorme Anforderungen an den Aufbau eines umfangreichen Anwendungssystems zur Unterstützung der effizienten Speicherung, Analyse, Verwaltung, Interaktion und Visualisierung der Daten gestellt. Die bestehende kostenlose 3D-Geodatenbank-Lösung „3D City Database“ (3DCityDB) entsprach bereits teilweise diesen Anforderungen, indem sie zwar eine umfangreiche Funktionalität für den Umgang mit den Standard-CityGML-Datenmodellen, jedoch keine Unterstützung für CityGML-ADEs bietet.

Die Schlüssel motivation für diese Arbeit ist es, einen zuverlässigen Ansatz zur Erweiterung der bestehenden Datenbanklösung zu entwickeln, um das effiziente Management, die Visualisierung und Interaktion großer Datensätze beliebiger CityGML-ADEs zu unterstützen. Der Schwerpunkt liegt zunächst auf der Beantwortung der Schlüsselfrage, wie man das relationale Datenbankschema dynamisch erweitern kann, indem die XML-Schemadateien der ADE analysiert und interpretiert und anschließend dem entsprechende neue Datenbanktabellen erzeugt werden. Auf Grundlage einer umfassenden Studie verwandter Arbeiten wurde ein neues graphbasiertes Framework entwickelt, das die typisierten und attribuierten Graphen zur semantischen Darstellung der objektorientierten Datenmodelle von CityGML-ADEs verwendet und anschließend Graphersetzungssysteme nutzt, um eine kompakte Tabellenstruktur zur Erweiterung der 3DCityDB zu generieren. Der Transformationsprozess wird durch die Anwendung einer Reihe feingranularer Graphersetzungsregeln durchgeführt, die es Benutzern ermöglicht, die komplexen Mapping-Regeln einschließlich der Optimierungskonzepte aus der Entwicklung des 3DCityDB-Datenbankschemas deklarativ zu formalisieren.

Der zweite wesentliche Beitrag dieser Arbeit ist die Entwicklung eines neuen mehrstufigen Systemkonzepts, das auf CityGML und 3DCityDB basiert und gleichzeitig als eine komplette und integrative Plattform zur Erleichterung der Analyse, Simulationen und Modifikationen der komplex strukturierten 3D-Stadtmodelle dienen kann. Das Systemkonzept enthält eine zusätzliche Anwendungsebene, die auf einem sogenannten „App-Konzept“ basiert, das es ermöglicht, eine leichtgewichtige Applikation bereitzustellen, die eine gute Balance zwischen der hohen Modellkomplexität und den spezifischen Anwendungsanforderungen der

Endbenutzer erreicht. Jede Applikation lässt sich ganz einfach mittels eines bereits entwickelten 3D-Webclients aufbauen, dessen Funktionalitäten über die effiziente 3D-Geo-Visualisierung und interaktive Exploration hinausgehen und auch die Durchführung kollaborativer Modifikationen und Analysen von 3D-Stadtmodellen mit Hilfe von der Cloud-Computing-Technologie ermöglichen. Dieses mehrstufige System zusammen mit dem erweiterten 3DCityDB wurde erfolgreich in vielen praktischen Projekten genutzt und bewertet.

## Chapter 1 Introduction

### 1.1 Motivation

In the past years, due to the increasing complexity of the city systems resulting from the growing urban population, more and more authorities like national and state mapping agencies worldwide have been motivated to create 3D spatial data infrastructures for monitoring the physical cities to enhance the interoperable data access among different sectors and organizations. The key concept is to link the relevant identifiable urban entities with the heterogeneous socio-economic and environmental data like energy performance indicators, pollution measurements, and air quality data etc. to provide an extensive information basis for performing a range of analyses and simulations (cf. Zheng et al. 2014). For example, it allows facilitating the decision-making processes in the energy and environmental sectors to increase energy efficiency and reduce harmful emissions respectively. Unlike using the traditional 2D spatial information, most of these analyses and simulations can highly benefit from the introduction of the third spatial dimension since many indicators can be computed or derived fully automatic from the 3D models. For example, the energy demand estimation for a building can be performed by calculating its 3D solid volume and outer surface areas, and the potentials for solar energy production can be estimated by analyzing the shadow casting phenomena in 3D (cf. Kaden 2014, Chaturvedi et al. 2017). Moreover, the shapes and the topological relations of the urban objects can also be visualized in a highly intuitive and natural way using a 3D scene which allows human operators to interactively explore the details about the city objects to check the data quality against the real-world.

***Semantic 3D City Models:*** The term ‘semantic 3D city model’ originates from the field of urban information modelling with the aim of reaching a common understanding of the urban phenomena within one framework. A semantic 3D city model is typically created with a clear decomposition of the relevant real-world objects into meaningful types like buildings, roads, railways, terrain, water bodies, vegetation and bridges with regards to their 3D geometry, topology, semantics, and appearance. It allows, unlike the traditional 3D graphics models, describing hierarchically structured 3D city objects based on their thematic attributes as well as spatial and semantic interrelationships (cf. Kolbe 2009). Semantic 3D city models can, hence, be a good basis for performing complex simulations and analyses etc., far beyond pure 3D visualization, since most urban applications do not just require data about the 3D geospatial characteristics but also semantic information (cf. Biljecki et al. 2015). In order to achieve interoperable access to semantic 3D city models, the international organization Open Geospatial Consortium (OGC) issued the CityGML standard, which has been designed as an open and universal information model for describing and exchanging semantic 3D city models. It facilitates the efficient management, exploration, and analysis of the heterogeneous 3D urban information over different users and applications. Today, more and more countries and cities worldwide are creating and maintaining their semantic 3D city models based on the CityGML standard to establish an integrative information backbone for accelerating their 3D spatial infrastructures (cf. Gröger et al. 2012).

***Extendable Semantic 3D City Models:*** Although semantic 3D city models can represent the most relevant 3D topographic feature types along with their relevant thematic and spatial

properties, additional feature classes or extra attributes are usually required to be added to the existing 3D city model for performing domain-specific analysis or simulations like energy demand calculations, utility network analysis, and noise propagation simulations. To resolve this issue, semantic 3D city models must be extendable such that their default feature catalogue can be dynamically enriched with additional feature classes from different application domains. Besides, since 3D city objects typically have well-defined identifiers which are usually kept stable throughout the lifetime of the referenced real-world objects, extra spatial and thematic properties can be dynamically attached to the existing feature classes. The creation and maintenance of such extendable semantic 3D city models can be realized by means of CityGML, which provides a flexible and systematic extension mechanism called “Application Domain Extension (ADE)”. It allows third parties to dynamically extend the existing CityGML definitions with additional domain-specific data models in a modular fashion, and can guarantee interoperable model interpretations across various application domains. For instance, it is possible to combine the building information from IFC with the urban information in CityGML to reach an Geo-BIM integration (cf. de Laet & van Berlo 2010) .

***Balancing City Model Complexity and User Needs:*** The introduction of additional extensions to the default 3D city model can inevitably result in an extra level of complexity on the entire model structure regarding the mixed thematic, geometric, and topologic information. This can bring both advantages and disadvantages: On the one hand, the extended 3D city models are enriched, and the datasets from different application domains are compatible with each other by sharing a common interface. On the other hand, the application users from various sectors normally just need to access a subset of the entire 3D city model to accomplish their domain-specific tasks, which, however, could be hindered by the barrier resulting from the high complexity of the underlying data models. Thus, a systematic approach is strongly needed in practice to reach a good balance between the high city model complexity and the specific user needs by means of a platform which should satisfy, amongst others, the following requirements.

- The complex 3D city models must be efficiently managed within a database serving as a central repository which should be flexible for handling any additional domain-specific model extensions. In addition, the semantics as well as the dependencies of the model structures shall also be reflected in the database for simplifying the data access and maintenance.
- The platform should additionally provide an intuitive user interface along with a simple data view that allows users to easily access and interact with the 3D city models via an efficient 3D visualization to perform data exploration, analysis, and simulations in a platform-independent environment.
- This platform shall allow the reliable credential management to secure the data that will be accessed by multiple application users. The users with different levels of authorization have to be assigned with the respective data access rights to facilitate the collaborative work on the same datasets.

To the best knowledge of the author, none of the existing application platforms on the market can fully meet the above-mentioned requirements. Besides, the conceptual solutions for

reaching these purposes have also rarely been discussed in literature in the past years. Based on this background, a novel approach has been developed in the context of this thesis and the related details are elaborated by examining and answering the following research hypotheses and questions.

## 1.2 Research Hypotheses and Questions

**Question 1:** Which technologies and standards are important for realizing the efficient management, interaction, and visualization of semantic 3D city models using a computer system?

**Question 2:** What are the key aspects for realizing the efficient storage and management of large and complex-structured semantic 3D city models regarding the data modelling and software implementation?

**Question 3:** How to implement the extensions to the semantic 3D city model according to the CityGML standard and then develop a dynamically extendable database for dealing with these extensions efficiently?

**Question 4:** Compared to the existing solutions, is there any further advanced approach that allows the automatic derivation of a compact relational database model from a CityGML extension which is structured as a complex object-oriented data model?

**Question 5:** How to realize the efficient visualization of large semantic 3D city models along with their extensions in a web browser, where users are able to explore the data model information interactively?

**Hypothesis:** It is possible to develop such an application system that supports management, visualization, and interaction of extendable semantic 3D city models, and at the same time reaches a good balance between the high model complexity and specific user needs. This balance can be evaluated against the criterion, whether end users who are not GIS experts can also access complex 3D city models and accomplish various domain-specific analysis and simulation tasks.

## 1.3 Organization of the Thesis

According to the research hypotheses and questions outlined in the previous section, the rest of this thesis is organized as follows:

**Chapter 2** introduces several technologies, concepts, and standards which are relevant for realizing the efficient management and visualization of 3D geospatial data. Included are Geographic Information System (GIS), Spatial Database System (SDBMS), Cloud Computing, Geospatial Data Modelling together with the ISO 19100 standard family, 3D Graphics Visualization, and Digital Virtual Globes, which all together form the theoretical foundations of this thesis.

**Chapter 3** firstly reviews the relevant concepts and features of the international standard CityGML which provides a common definition of the basic entities, attributes, and relations of a semantic 3D city model. Additionally, the relevant relational database modelling approaches are examined, based on which a complete database solution called 3D City

Database (3DCityDB) were developed which came with a set of software tools allowing to import, manage, analyze, visualize, and export semantic 3D city models according to the CityGML standard. The technical details of these software tools are illustrated in detail in the last subsection of this chapter.

**Chapter 4** mainly focuses on the key approaches for the efficient management of the domain-extended semantic 3D city model according to the CityGML's extension mechanism ADE. In this context, the relevant concepts of the CityGML ADE have been first reviewed together with a modular 3DCityDB database structure for dealing with CityGML ADEs. Particularly, a new graph-transformation-based approach is presented in detail which has been developed for automatically generating a compact 3DCityDB-compliant relational database schema from a given ADE application schema constructed with an object-oriented data structure. Moreover, a prototypical implementation of the concept solution is accomplished and illustrated along with an application example.

**Chapter 5** presents the relevant approaches for realizing the interactive 3D exploration of large semantic 3D city models to facilitate users to access the city model information intuitively. These approaches include the methods of generating the dedicated 3D visualization models from the semantic 3D city models for realizing the efficient data streaming and 3D geo-visualization on the web. Regarding the rich data exploration and interaction of the individual 3D model objects, the Cloud Computing technology and the Cesium Virtual Globe are utilized to develop a 3D web client which can be used to build a simple and user-friendly 3D web viewer for a GIS application. The concepts and technical implementation of this 3D web client are illustrated in detail along with a number of practical application examples.

**Chapter 6** proposes a new framework with a multi-level system architecture for facilitating the employment of complex extended semantic 3D city models in domain-specific applications. In this chapter, the conceptual idea of this approach has been first presented which is based on an app-concept allowing to build a simple application according to the specific application requirements to reach a sophisticated balance between the city model complexity and the user needs. Additionally, the technical details of this developed system together with an extended 3D web client are presented which is mainly realized on top of the research and development results achieved from the former chapters. Finally, two application examples are given to demonstrate the applicability of the developed system for the practical use cases.

**Chapter 7** first draws the conclusions about this thesis with respect to the stated motivations and the outlined research hypothesis and questions. In the subsequent subsection, the main contributions and efforts of this thesis to a range of fields like scientific research, practical applications, academic teaching and practical training, as well as international standardization are summarized according to the results obtained from this research and implementation work. Finally, the potential possibilities for improving the developed approaches and software applications are identified which needs to be further investigated in future research in order to accelerate the enterprise management, interaction and visualization of semantic 3D city models.

## Chapter 2 Theoretical Background

This chapter is intended to give an overview of the relevant technologies and standards which cover the key aspects of the research work presented in the subsequent chapters. First, a brief explanation of the technology *Geographic Information System (GIS)* is given which forms the fundamental backbone for the entire development work of this dissertation. In the subsequent section, the *Spatial Database System* acting as one of the most important components in GIS environments is illustrated as it is considered powerful means for realizing the efficient storage and management of 3D geospatial data. Concerning the data exploration and interaction of semantic 3D city models in practical applications, a novel approach has been proposed and developed in the context of the thesis. It allows to construct a user-friendly application framework which is based on the coupling of Cloud-based services with a Web-based GIS implementation. Therefore, for the sake of clarity, a brief explanation of *Cloud Computing* is given in this chapter. Additionally, the standard approach for *Geospatial Data Modelling* has been reviewed in order to facilitate understanding the basic ideas for the development as well as for the management of semantic 3D city models with their dynamic extensions. Finally, the technologies i.e. *3D Graphics Visualization* and *Digital Virtual Globe* are illustrated in the rest part of this chapter, since they play an important role in building a 3D viewer for supporting high-performance interaction, exploration, and visualization of semantic 3D city models in a GIS application.

### 2.1 Geographic Information System (GIS)

The term *Geographic Information System (GIS)*, also called *Geographical information System* stands for those kinds of computer systems that support for acquiring, storing, managing, analyzing and displaying geospatial data (cf. Chang 2006). As this definition implies, the ability of handling geospatial data information separates GIS from other information systems and hence brings GIS to a wide variety of application fields like urban planning, emergency analysis, resource management, and natural hazard assessment etc. In recent years, with the advancement of computer technology, more powerful and cost-effective hardware have dramatically accelerated the evolvement of GIS for managing large geospatial data on a variety types of platforms and devices such as personal computers and mobile devices. Like with other computer application systems, a GIS environment is typically made up of a set of system components which can be hierarchically constructed to accomplish certain application tasks. According to the literature (cf. Vijlbrief & van Oosterom 1992, Jung 2008), the typical system architecture is shown in Figure 1.

The fundamental component of GIS is the geospatial data which can be created by digitizing paper maps or captured by means of geospatial technologies like remote sensing, surveying, and photogrammetry to integrate such various measured data into a common platform. These data shall be organized in a sufficient data structure i.e. vector or raster, and also with an appropriate spatial reference system i.e. geographic coordinate system, projected coordinate system or a compound coordinate system which is a combination of a two-dimensional projected coordinate system with a vertical coordinate system (Kothuri et al. 2011). All these geospatial information shall be maintained in such a database system that must be enhanced with additional spatial capabilities allowing for efficient storage of geospatial data. In

addition, since the database systems are usually accessible over the network by applications, interoperable access to a common database is guaranteed for the data management i.e. update, deletion, and enrichment. Moreover, these kinds of spatially-enhanced database systems usually offer powerful functionalities like performing queries and analysis on spatial data which can be invoked in various spatial analysis and simulations like buffer analysis, viewshed calculations, spatial interpolations, path and network analysis (cf. van Oosterom & Vijlbrief 1994, van Oosterom et al. 2002). The calculation results along with the original geospatial data can be displayed on a client application using a 2D/3D map viewer which allows GIS users to visually explore and inspect the respective spatial and thematic information as well as to operate the GIS functionalities in an interactive way.

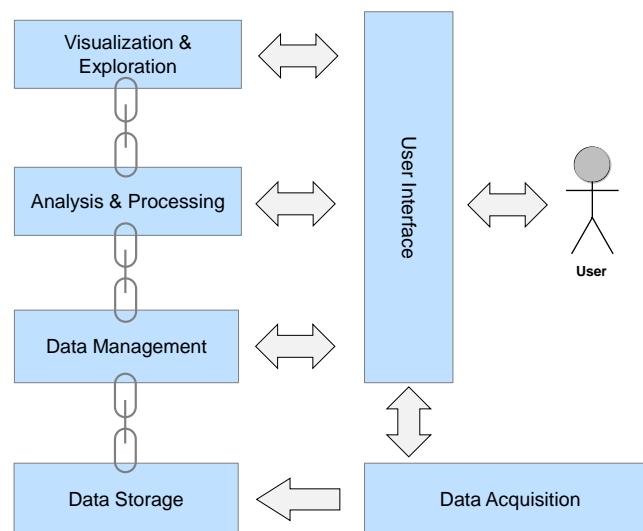


Figure 1: Relevant components of a typical GIS environment (cf. Jung 2008)

Depending on the platforms on which the GIS client applications are run, GIS can be roughly categorized into three types, namely desktop GIS, mobile GIS, and Web GIS (Amirian & Alesheikh 2008), each of which can be further specialized into 2D and 3D variants according to the visual dimensions supported the map viewer. Most GIS client applications used in industrial and academic fields come under the desktop GIS category which are usually able to provide the full functionalities ranging from the data creation and modification up to geospatial analysis and processing using an integrative user interface. One of the representative commercial GIS products is ESRI ArcGIS, which provides rich GIS functionalities and has been widely used in research and production in the past years. A number of additional free and open source desktop GIS application such as QGIS, gvSIG, OpenJUMP GIS and GRASS etc. also provide sophisticated GIS functionalities and have reached a certain market share in many application domains. Compared to the desktop GIS, a mobile GIS application is normally a light-weight version of its desktop-based counterpart and has a simplified GUI layout and functionalities to satisfy the shranked screen size of the mobile devices. The main advantage of the mobile GIS is that users can carry the mobile devices to anywhere and hence is useful for acquiring geospatial data on-site. However, most mobile and desktop GIS applications are platform-dependent and can only run on some certain operating systems. In addition, the GIS applications must be installed on the devices beforehand which require a certain level of access permission to the operating systems of the devices. This will strongly restrict the use of GIS applications in a wide scope since normal



users are usually not granted with such privilege to install a software on corporate devices. This drawback can be overcome by using Web GIS. A Web-based GIS application typically use web browsers at the client side which allow to reach the platform-independent purpose. With the latest version of Hypertext Markup Language HTML5, it is possible to create powerful 2D and 3D GIS client applications by means of the HTML5 features e.g. canvas element, multi-threading (Web workers), offline data storage, and Geolocation (cf. Faulkner et al. 2017). The communication between client and server in Web GIS usually relies on HTTP specifications to transfer data requests and responses via the OGC standards e.g. Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS), and Web Processing Service (WPS) (cf. Evans & Sabel 2012). Nowadays, an increasing number of web-based GIS applications have been developed by GIS vendors and have been successfully employed in various application fields.

## 2.2 Spatial Database System

As mentioned in the previous section, the employment of database technology plays the central role in a GIS application for the efficient management of geospatial data in a consistent repository when dealing with large-size data and/or a large number of concurrent users (cf. Shumilov et al. 2002). In general, a database system (DBS) typically consists of two components, namely database and database management system (DBMS). While the database is a collection of datasets, the DBMS is a software which allows easy access and organization of the data stored in the database. A spatial database system (SDBS) is an enhanced version of the ordinary DBS by extending the capabilities of the database to support high-performance storing, querying, and processing of geospatial information. It provides a set of spatial data types, in addition to the primitive data types i.e. string, floating point number, integer and date etc. for storing different geometry types such as points, lines, polygons, solids, or a collection of them (cf. Güting 1994). In addition, a spatial database may also be able to store raster data, triangulated irregular networks (TINs), as well as massive point cloud objects (Kothuri et al. 2011), all of which can be represented with 2D or 3D coordinates based on real-world coordinate reference systems.

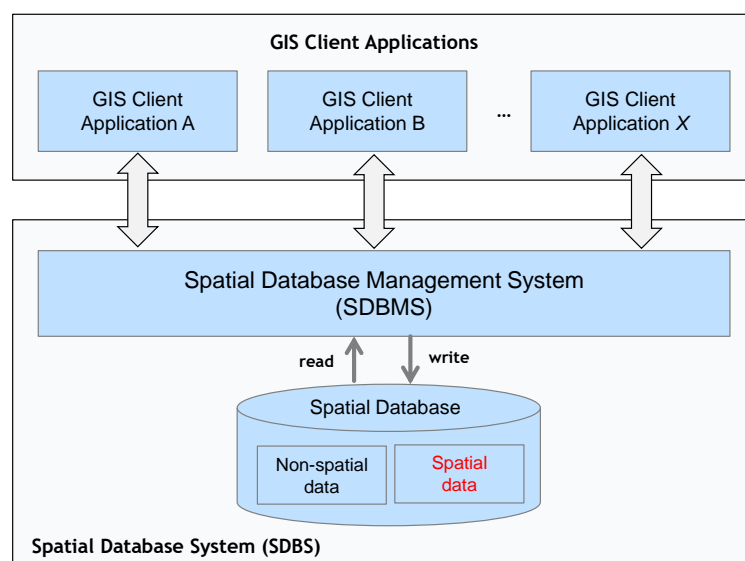


Figure 2: Application structure based on a spatial database system (cf. Brinkhoff 2005)

Since the SDBS is an extension of the ordinary DBS, all functionalities of a DBS can be made full use of in practical applications. For example, when the geospatial data stored in the database are accessed by multiple users, a locking mechanism allowing for the concurrent access is supported for preventing inconsistent results from data transactions. In addition, the management of user access permissions is also supported by the DBMS and allows database administrators to assign users different levels of access permissions for ensuring data security and consistency. For example, a read-only access right can be granted to public users such that the risk of altering the sensitive database data can be avoided. In order to speed up the access performance when dealing with large data, multiple queries sent from different client applications can be concurrently processed by the DBMS (cf. Figure 2). In addition, various user-defined operation rules can also be implemented using the trigger mechanisms provided by the database which allow to capture the events of changes made to the stored data and programmatically propagate the update to the other data in the same database automatically. Moreover, some databases products e.g. Oracle also support version management mechanisms which allow to log and trace the history of the data modifications and roll back the data contents to an earlier status of the database at a certain timestamp (cf. Beauregard et al. 2009).

Compared to conventional DBS, another key feature of SDBS is the capability of creating spatial indexes to boost the performance of the spatial operations, i.e. querying and processing of large geospatial data. For example, some GIS analyses usually perform spatial queries to answer the question like ‘which geospatial objects are contained within a certain spatial extent i.e. a 2D bounding box’. A quick response of such query is especially important for the ad-hoc interaction with the geospatial data for facilitating real-time GIS applications. In addition, spatial indexes can also facilitate spatial operations such as spatial joins which are typically used for creating an abstract view of all those objects whose locations or extents are spatially related. There are also a number of further spatial functions and operations that can be significantly sped up using spatial indexes. Thus, creating spatial indexes is one of the first important steps when handling and storing large geospatial datasets in a SDBS (cf. Kothuri et al. 2011).

The basic idea of creating spatial index is similar to conventional database indexes like the B-tree which tries to order all data records in a hierarchical structure for speeding up the data searching process. For handling spatial data with higher dimensions, a number of spatial indexing methods have been developed which are based on different spatial structures e.g. Grid, quad-tree, k-d tree together with the respective searching algorithms. However, most of them are only well-suited for indexing points and are not sufficient for geometries like lines, polygons, and solids (Kothuri et al. 2002). Therefore, most database products like Oracle Spatial and PostgreSQL/PostGIS implement the so-called R-tree approach which is a more powerful indexing method with respect to spatial queries on multi-dimensional spatial objects. It has a B-tree like structure and is a balanced search tree whose leaf nodes must be at the same height and each has a minimum bounding rectangle that spatially contains the minimum bounding boxes of the referenced data objects. The non-leaf nodes are linked with their child nodes in the similar manner (cf. Guttman 1984).

A simple example of the R-tree implementation is presented in Figure 3 where the minimum bounding boxes (R8 – R19) of 12 polygons are distributed in a 2D plane space and are

spatially intersected, disjoint, or overlapped with each other for representing the topological relationships which have frequently occurred in the real-world objects. When searching those polygons that enclose a given point object, the conventional method without using spatial index has to iterate through all the polygons to check whether the point is intersected with the visited polygons. This way, the checking process must be performed 12 times and will consume linearly increasing calculation time when the number of polygons is growing  $O(n)$ . In comparison, the R-tree index allows the computer to rapidly find out the target polygon by traversing the tree from its root node to the leaf nodes based on their spatially correlated bounding boxes. In this case, only three calculation steps are needed with a lower computational complexity  $O(\log_M n)$ , where  $M$  is the minimum number entries in one node.

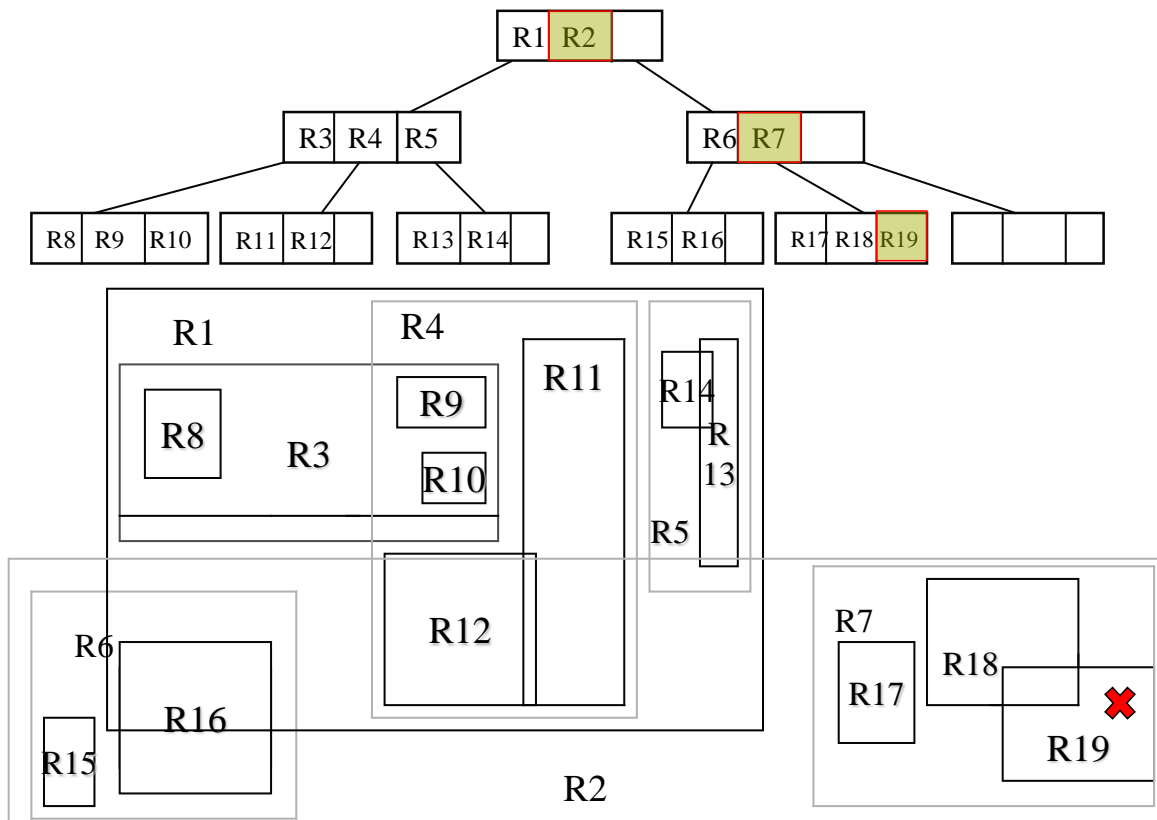


Figure 3: An example of R-Tree spatial index (cf. Guttman 1984)

Nowadays, a number of open-source and commercial database products with enabled spatial extensions have been developed which can be roughly categorized into two types namely, non-relational and relational databases, according to the data structure managed within the database (cf. Tauro et al. 2012). The former can be further classified into object-oriented database, document-oriented databases, and graph databases etc., which are shortly explained in the following.

- **Object-oriented database:** it is also called object database where the stored content information are organized by following an object-oriented structure which is commonly used in object-oriented programming languages (cf. Atkinson et al. 1992). Thus, the relationships between objects such as inheritance and associations can be fully represented in the database, and it allows to interact with the stored objects using object-oriented programming languages in a straightforward fashion. Supported

database products are Objectivity/DB, ObjectStore, and InterSystems Caché etc. (cf. Pratiksha & Pursani 2014).

- **Document-oriented database:** within this database, the data objects are represented as a document which contains the object attributes and their values being structured as JSON or XML documents for storing hierarchically data. Thus, the well-known XML database belongs to the category of the document-oriented database. Since the data are stored as single document in the database, the database model is schema-less and can be easily accessed by application programs without needing to know the database schema beforehand (cf. Boicea et al. 2012). In addition, compared to relational databases, the update on the stored object data can be easily done by just editing the corresponding document. There is no need to care about various constraints on the related tables to ensure database consistency and integrity. A representative document-oriented database product is “MongoDB” which is a spatially-enhanced XML database management system and has been proven to be a sophisticated database solution for storing GML-compliant geospatial data (cf. Mao et al. 2014).
- **Graph database:** As the name implies, the underlying data model of a graph database is represented using a graph structure where a graph node is normally used for representing objects, whereas graph edges represent the relationships between the respective objects. Each node can be assigned with an arbitrary number of attributes along with their values formed in a so-called “key-value” data structure (cf. Vicknair et al. 2010). With the graph structure, the complex data models having tree or graph structures can be directly mapped onto a graph-based database schema. This way, the syntactical structure of the original data model can also be well kept in the database which allows rapid query, processing, and comparison of the data (cf. Nguyen 2017). A representative database product is ‘Neo4J’ which is an open-source graph database management system and is also featured with sophisticated spatial capabilities for handling geospatial data.

Although the above-mentioned non-relational databases are becoming more and more popular in many application fields (cf. Ordonez et al. 2010), they are currently still limited in their capabilities and performance of performing certain kinds of spatial operations and coordinate transformations which are of great importance for the enterprise use in GIS applications (cf. Agoub et al. 2016). Therefore, the spatially-enhanced relational database systems such as the commercial Oracle with ‘Spatial’ extension and the open-source PostgreSQL with ‘PostGIS’ extension etc. are nowadays predominantly employed in GIS world due to their extensive abilities in handling geospatial data.

### 2.3 Cloud Computing

One of the clearest definitions of Cloud Computing was given by (Armbrust et al. 2010):

*“Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services”.*

As the definition indicates, the Cloud Computing technology offers an Internet-based solution that allows system administrator and application developers to just focus on the business work without the need to build up the physical computer infrastructures, since such computing

resources can be managed and provided via a central service pool known as “Cloud”. It allows companies or developer users not only to minimize the resource and maintenance costs but also to speed up the development time of their business products. In order to enable on-demand access to the resources in the Cloud, the key concept behind the Cloud Computing technology is virtualization (cf. Figure 4), which means that a bundle of physical computers are grouped and shared by multiple virtual machines which are able to dynamically allocate the computing resources according to the actual usage demands. In this way, Cloud services can be provided with optimized usage efficiency of the hardware by avoiding the waste of unused resources (cf. Armbrust et al. 2010).

Cloud Computing can be classified into different types according to the accessibility of the computing resources. The first type is called ‘Public Cloud’ whose services are accessible to the public users over the Internet. If the Cloud is just used within an intranet like a company’s internal network, such type of Cloud is called ‘Private Cloud’. The combined version of the ‘Private Cloud’ and ‘Public Cloud’ is simply called ‘Hybrid Cloud’. In the case of a public Cloud, the Cloud services can be sold based on a “pay-as-you-go” manner which is similar to the case for energy and telephone services. In this way, the Cloud users just need to pay the costs of the computing resources based on their actual needs and can dynamically adjust the payment when there is an increase or decrease in their usage demand (Fernando et al. 2013).

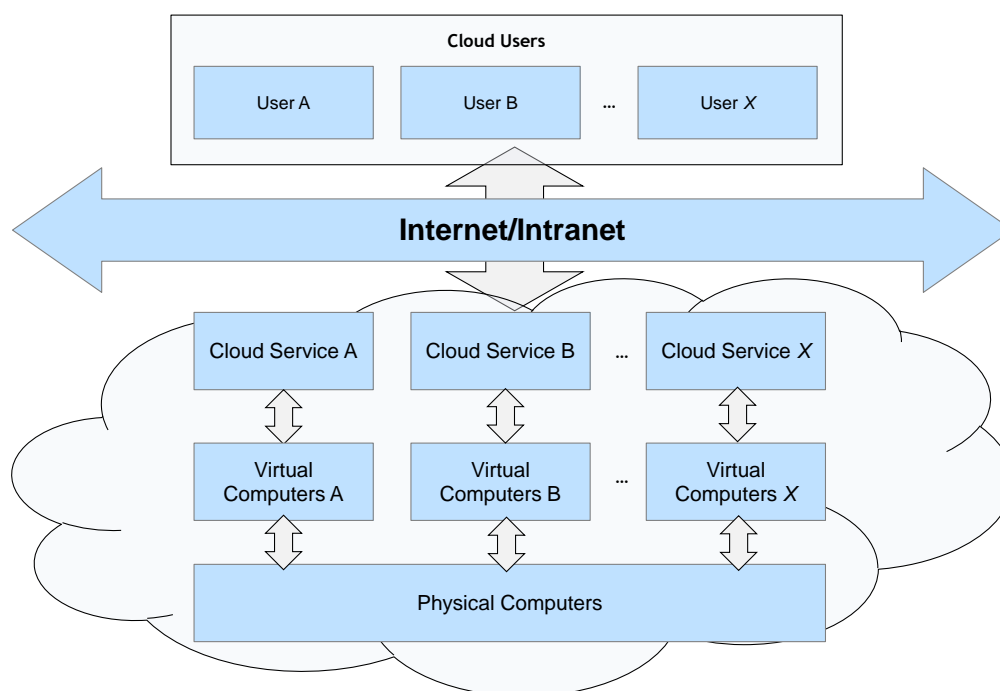


Figure 4: Conceptual system architecture of Cloud Computing (cf. Buyya et al. 2008)

With the growing popularity of Cloud Computing in the IT field, different types of cloud services are provided by Cloud Service providers to meet the various user needs. Depending on the application level, the Cloud services are commonly categorized into three types (cf. Buyya et al. 2008), namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS):

- **Infrastructure as a Service (IaaS)** offers the highest level of the control of the infrastructure components including hardware, servers, and storage etc. which are managed and monitored by the IaaS providers. It usually provides the functionalities

like automation of administrative tasks, dynamic scaling, desktop virtualization, and policy-based services (cf. Bhardwaj et al. 2010). The IaaS services can be charged based on the actual use time or the amount of consumed virtual machine space. Thus, the IaaS is very cost-effective for the Cloud users like a company who may want to test its application on a temporary platform and later switch to an in-house deployment if the software prototype has successfully past the tests. To date, the leading IaaS providers include e.g. Amazon Elastic Compute Cloud (EC2), IBM SoftLayer, Google Compute Engine etc.

- **Platforms as a service (PaaS)** refers to a ready-made service, which includes not only the low-level infrastructure resources but also the deployment platforms and runtimes for running applications. It releases Cloud users from managing and controlling the underlying cloud infrastructure like hardware, network, operating systems, and databases etc. and allows them to just focus on the development of their applications without the need to worry about things like resource scalability, software maintenance, and security. This makes it possible to realize a rapid development and deployment of software applications with lower costs since the required platforms can be fully managed by the PaaS providers. Typical PaaS-based Cloud Services are e.g. Microsoft Azure, Google App Engine, and Apache Stratos.
- **Software as a Service (SaaS)** provides complete services or software applications which are run and managed by the Cloud providers. In general, SaaS users just need to know how to use and operate the provided software and do not have to concern how the platform is managed or how the underlying infrastructure is constructed. A simple example of a SaaS application is the web-based file hosting service Google Drive where users can upload and share their document files over the Cloud without having to maintain the underlying server configurations, hardware storage, or operating systems that the software service is running on. Besides Google Drive, other representative SaaS-based Cloud services include Dropbox, Microsoft OneDrive, and Google Apps etc.

The relationship between these Cloud services and products are illustrated in Figure 5.

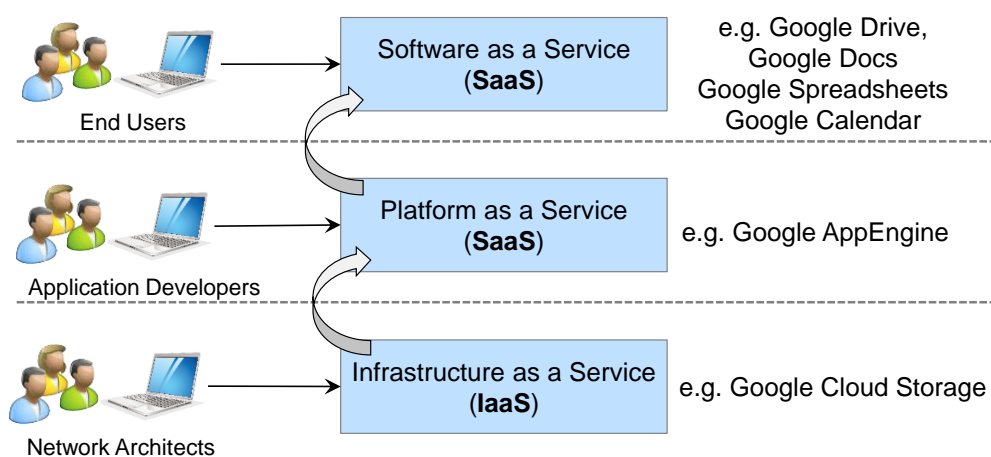


Figure 5: An example of the three types of Cloud-based services using the Google Cloud ecosystem

In the past years, the Cloud Computing technology has also revolutionized the way of deploying GIS applications. While the software components required in a traditional GIS

solution are typically installed or deployed on physical computers, the Cloud-based GIS shifts its system components including spatial databases, web servers, and mapping applications etc. to the Cloud, which allows to build an advanced GIS infrastructure but with relatively lower cost. Besides, since the Cloud-based system can eliminate the work of deploying the complex in-house hardware and software, GIS developers only need to focus on the application logics and do not to have to master the specialized skills for maintaining the complex GIS environment. Moreover, the Cloud-based service along with its authorization mechanism also allows GIS customers to perform secured data access and accomplish collaborative GIS tasks (cf. Bhat et al. 2011).

Nowadays, one of the most popular Cloud-based GIS solutions is the ESRI's ArcGIS Online which provides a complete platform allowing users to access, create, edit, and analyze geospatial data on the Web (cf. ESRI 2017). Alternatively, it is also possible to construct a GIS environment using a combination of the other existing Cloud service products. For example, GIS developers are able to use the Google App Engine to build a web-based mapping application for viewing and analyzing geospatial data (cf. Karimi et al 2011). At the server side, the Amazon EC2 can be utilized to deploy web servers on different kinds of operating systems like Windows and Linux for hosting the geospatial datasets in databases or file systems that can be accessed via elastic IP addresses over the Internet. When the client applications are reading and loading the geospatial data, the capacities of the web server can be dynamically adjusted according to the actual usage demands and allow to guarantee the data storage and retrieval performance. Moreover, the Google Sheets can be used as a light-weight engine for realizing the location-based GIS visualization and exploration (cf. Gonzalez et al. 2010).

## 2.4 Geospatial Data Modelling

As mentioned in the first subsection, geospatial data play a fundamental role in GIS applications for the representation and analysis of real-world phenomenon. For instance, the vector data such as points, lines, polygons, and solids can be used for quantitatively describing the locations and shapes of various geospatial objects and allow for calculating the relevant morphological characteristics like length, area, and volume. In addition, it is also possible to utilize vector-based representations to explicitly describe the topological relationships between geospatial objects (cf. Maffini 1987). For example, if two building objects are adjacent in the horizontal direction, this topological relationship can be explicitly modeled using one polygon or multi-polygon geometry to represent the shared wall surface which shall be referenced by the two respective building objects. Compared to vector data, raster data is very suitable for representing evenly distributed geospatial phenomenon such as digital terrain model which can be approximated as a georeferenced raster grid where each cell has a numeric value to represent the elevation height of the corresponding geographic position.

In general, all these kinds of geospatial data along with the related thematic attributes must be organized within a suitable data structure that can be efficiently handled within GIS applications. A common approach is called feature-based modelling, which represents real-world entities as features associated with their spatial and thematic attribute as well as topological relations (cf. Chang 2006). In the past years, a number of GIS vendors have

developed their own geospatial data models in the sense of heterogeneous application schemas in their GIS products. However, most of these application schemas are platform-dependent and hence usually incompatible with each other while maintaining the data in different systems (cf. Kutzner 2017). This will strongly hinder the interoperable dissemination of geospatial data for a wide range of GIS applications since the different model interfaces must be implemented by the third parties and may result in high development costs for realizing the schema transformation. Thus, an open and standardized approach for the modelling of geospatial data at a higher abstract level becomes more and more important in the GIS field in order to provide a standardized and platform-independent data model that can be interpreted by different GIS applications using a common interface (cf. Figure 6).

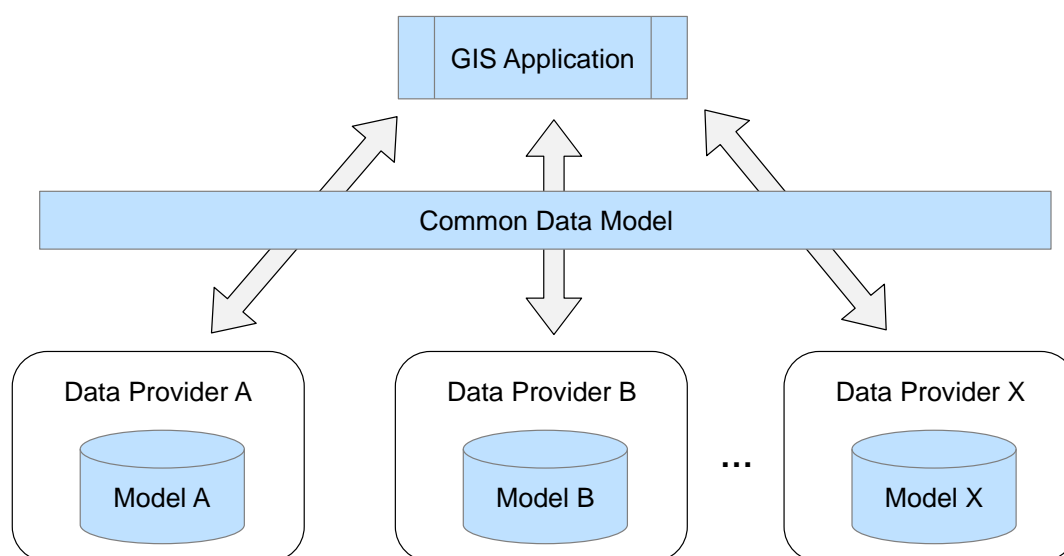


Figure 6: Conceptual idea of the interoperable data dissemination using a common data model in GIS

To reach this goal, a systematic way for developing geospatial data models is required. The common approach of developing data models in software and systems engineering is the so-called “Model Driven Architecture (MDA)” issued by the Object Management Group (OMG). The MDA provides a standard workflow for facilitating the development of software and data models using Computer-Aided Software Engineering (CASE) tools. The key idea behind this approach is that an abstract platform-independent model (PIM) representing the underlying conceptual models shall be first created at the earlier stage of the development phase and then be implemented for different target application platforms at a later stage by transforming it into the corresponding platform-specific models (PSM) such as XML schema and database schema, which can be directly handled in the concrete application systems (cf. Gasevic et al. 2006).

The main advantage of this approach is that the definition and development of data models can be purely done at the conceptual level without concerning the concrete implementation for specific platforms. The PSMs can be automatically derived from the developed conceptual model using a model transformation tool at any time (cf. Figure 7). This allows to minimize the development and maintenance costs for the entire development lifecycle. In order to ensure the unambiguity and consistency of the definition and description of model structures, the Unified Modelling Language (UML) was chosen as the standard modelling language for describing the platform-independent models since it comes with a rich set of graphic notations



and syntax allowing to visually represent complex model structures as well as to fully represent the respective semantics (cf. Siwik et al. 2010). Thus, the UML model is considered a good start point for the development of a new data or application model (cf. Zulkifli NA et al. 2014, Alattas et al. 2018). One of the existing tools supporting the MDA-based approaches is Enterprise Architect (EA) which is a powerful design tool for developing a variety of models like entity-relational models, relational database models, activity/progress models, and object-oriented models according to the different application requirements.

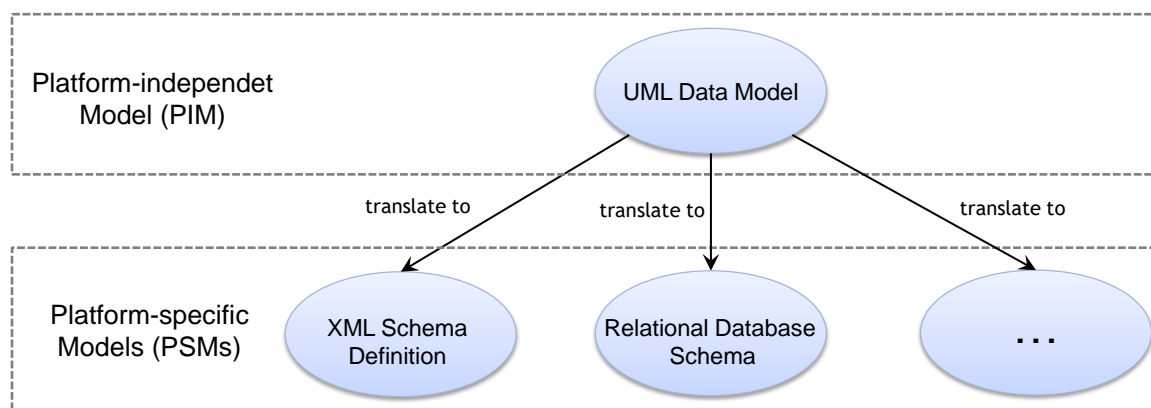


Figure 7: Relationship between PIM and PSMs in a Model-Driven Architecture

Nowadays, the combined use of model-driven approach and UML has become the standard procedure for developing geospatial data models in the GIS domain (cf. Kutzner 2016). The UML data models are commonly defined according to geospatial standards called “ISO 19100 standards family” issued by the Technical Committee (TC) 211 of the International Organization for Standardization (ISO). This bundle of standards comprises a series of specifications which jointly provide a general guideline for geographic information modelling regarding the definition and description of geographic phenomena in order to ensure the interoperability of data exchanges across different application platforms (cf. Faucher & Lafaye 2007). In total, over 40 ISO standards have been issued which cover most relevant modelling aspects, among other things, such as temporal schema (ISO 19108), metadata (ISO 19115), and coverages (ISO 19123) are explicitly covered.

One of the most important members of the ISO 19100 standards family is the ISO standard 19107 “*Spatial Schema*” specification (ISO 19107:2003) which is a conceptual schema for describing the geometric-topological characteristics of geographic features. Most relevant geometry as well as the topology models associated with spatial objects have been fully represented using class diagrams and package diagrams expressed in UML. The entire schema has been subdivided into several packages to jointly represent a variety of different geometry variants with respect to the ISO 19111 standard, which defines the conceptual schema for the description of spatial referencing by coordinates. For example, geometric primitives such as point, curves, planar polygons, and 3D solids have been explicitly modelled ranging from 0D to 3D. Complex structured geometries can be represented using a mixed collection of the provided primitive geometries and different aggregation concepts such as aggregates, complexes and composites are also supported which differ in terms of the topological relationships of the underlying geometry elements. In addition, the representation of a solid object is based on a so-called B-Rep approach which uses a set of polygons to

enclose the respective object's body. Moreover, holes in the solid's outer shell can also be represented using oriented surfaces.

Another important standard is ISO 19109 “*Rules for Application Schema*”, which specifies the relevant rules on modelling real-world features along with their spatial and non-spatial properties as well as their interrelationships such as generalization/specialization and association relationships like aggregations and compositions (ISO 19109:2005). It hence provides a standard metamodel framework for developing application schemas to facilitate the processing, analyses, simulation, and exchange of geospatial data in GIS applications. The main concept of the standard is the General Feature Model (GFM) with the notion of ‘Feature’, which is an abstraction of real-world phenomena to represent geospatial features such as building, rivers, parcels, and waters along with their semantic decomposition hierarchies. Each feature can have zero, one, or more spatial attributes whose values are restricted to geometrical and topological objects defined in the aforementioned ISO 19107 standard. This way, the coupling of ISO 19107 and ISO 19109 allows to extensively represent a feature object with respect to its spatio-semantic coherence (cf. Stadler & Kolbe 2007). For example (cf. Figure 8), a building's outer shell can be geometrically represented as a volume (solid), which consists of multiple polygons according to the ISO 19107 standard. This building can at the same time be semantically decomposed into thematic surfaces like wall surfaces and roof surfaces modelled based on the ISO 19109 standard. Both geometric and semantic representations can be coherently applied by linking the respective building components.

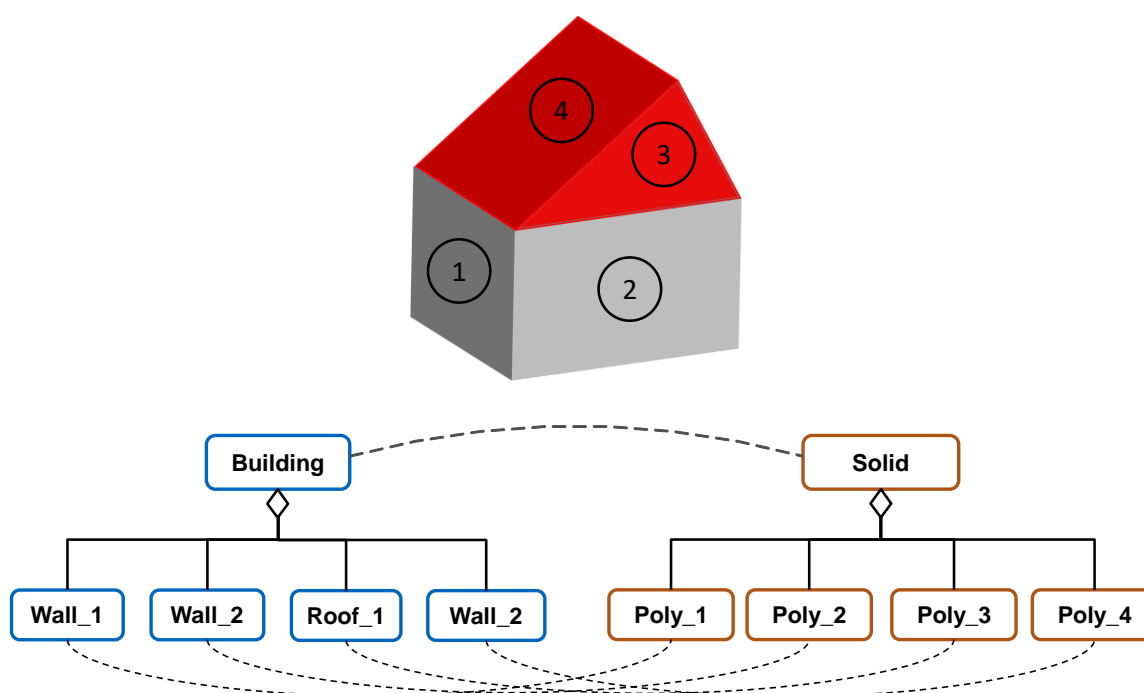


Figure 8: Spatio-semantic representation of a building based on ISO 19107 and 19109 standards

In order to provide an open and manufacturer independent framework for the development of geospatial data models based on the ISO 19100 standards family and MDA concept, the international specification “*Geography Markup Language (GML)*” has been jointly developed by OGC and ISO and was issued as ISO 19136 standard. It allows for semantically rich and object-oriented data modelling of the geo object on the application level (ISO 19136:2007).

The main GML model components like features, attributes, geometries, topologies, coverages, coordinate- and time reference systems are mostly drawn from the conceptual models defined in the ISO 19100 standard family and mapped to an XML-based schema (GML schema) according to the XML encoding conventions specified in the ISO 19118 standard (ISO 19118:2011). The GML schema hence serves as a standardized and fixed XML encoding of the conceptual models from the standards e.g. ISO 19107, ISO 19108, and ISO 19109 to increase interoperabilities and reduce ambiguities among applications. In addition, GML provides an extensive UML profile, which adheres to the conceptual models from the standards ISO/TS 19103 and ISO 19109 for developing ISO 19109 conformant UML application schemas. This UML profile complements the UML profiles of the two standards by introducing additional stereotypes and tag definitions to help transformation tools to automatically map user-defined UML application schemas to GML application schemas, which reference (import) the fixed GML schema. The corresponding encoding rules are based on the ISO 19118 standard and given in Annex E of the ISO 19136 standard. An overview of the relationships between the conceptual models and their encodings are shown in Figure 9.

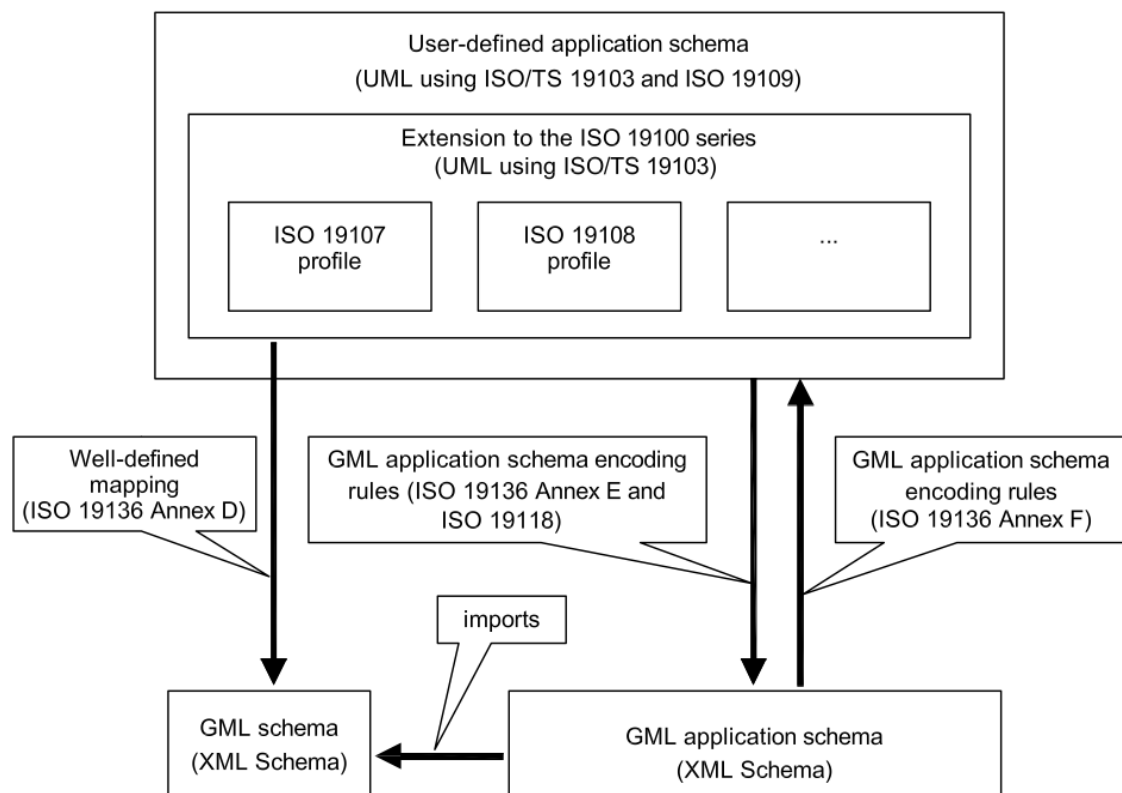


Figure 9: Relationship between the conceptual models from the ISO 19100 series and their GML encodings (ISO 19136:2007)

The UML profile specified in the ISO 19136 standard offers a set of UML stereotypes which can be used for the modelling of application-specific object types and elements:

- **<<Type>>** is used for modelling the distinguishable objects which shall have unique identifiers and can therefore be referenced by other objects. According to the GML specification, this stereotype corresponds to the GML class *AbstractGMLType* in the GML schema. In case that a class is declared with this stereotype in a UML model, this class will be encoded in the derived application schema as a global XML element

along with a global XML complex type being a subtype of the class *AbstractGMLType*. In addition, the class *AbstractGMLType* also contains a set of pre-defined attributes such as *GMLID* and *GMLID\_Codespace* which can be combined to serve as a unique identifier for the individual GML objects. A number of the standard GML types such as GML feature, geometry, topology, and coordinate reference system etc. are derived from this class.

- **<<FeatureType>>** can also be used for representing an identifiable and distinguishable real-world object and is conform to the abstract class ‘Feature’ defined in the ISO 19109 “*Rules for Application Schema*” standard. Basically, this stereotype corresponds to the abstract GML class *AbstractFeatureType* which is a subtype of the GML class *AbstractGMLType* and therefore inherits the relevant attributes like *GMLID* and *GMLID\_Codespace*. It also receives a number of additional attributes such as 3D spatial extent of a feature object, multiple names with different codespaces, and a feature description. In GML application schemas, each class defined with this stereotype shall be encoded as a global XML Element along with a global XML *ComplexType* being an extension of the GML type *AbstractFeatureType*. For example, the GML’s built-in class GML *Coverage* is defined by extending the *AbstractFeatureType*.
- **<<Enumeration>>** has a similar functionality compared to the Enumeration type used in the programming language like Java and C++. According to the ISO 19136 standard, a class defined with this stereotype should be encoded as a global XML Type *SimpleType*. Note that there exists an inconsistency between the UML model and the encoded GML application schema. Regardless of the data type specified for the Enumeration class in the UML model, the enumerated values are rigidly encoded with the restriction to the XML’s built-in type *xsd: string* according to ISO 19136 Annex E (cf. Table E.1, page 324). As a result, the enumeration value contained in an object or feature instance is always represented as a string although it may sometimes should be used for representing a numeric value.
- **<<CodeList>>** is similar to the **<<Enumeration>>** for enumerating a list of values and is per-default encoded as the XML *SimpleType*. The difference is that, in a CodeList, each enumerated value can be extended with additional values which are encoded using the XML’s annotation element *appinfo* for describing the details of the individual listed code value. In addition, when setting the tagged value “asDictionary” to “true” in the UML model, a CodeList can be alternatively implemented as an external file outside the GML application schema according to the GML’s Simple Dictionary Profile. The latter approach has been frequently chosen in practice since a CodeList can then be encoded in a formal XML structure, while the XML annotations like *appinfo* can contain any content which can sometimes not be validated against XML schema definitions.
- **<<DataType>>** is defined as a set of properties that lack identity (ISO 19136:2007). It has a similar functionality compared to the ‘Struct’ type introduced in the C/C++ programming language. The contained values can be defined with the simple primitive data types like *string*, *boolean*, *double*, *date*, and *integer* etc. as well as those complex data types that are defined using the stereotypes **<<Type>>**, **<<FeatureType>>**, and

<<DataType>>. In GML application schemas, the class defined with this stereotype shall be encoded as a global XML element along with a global XML *ComplexType*. In addition, a *DataType* can also be defined by extending another *DataType* using the UML's inheritance notation. In this case, XML-encoded *complexType* of the sub-*DataType* shall have an <Extension> element pointing to the super *DataType*.

- <<*Union*>> is a set of properties and it is similar to the <<DataType>> for holding a list of values or objects (ISO 19136:2007). In GML application schemas, it is also encoded as a global XML element together with a global XML *ComplexType*. The <<Union>> and <<DataType>> are mainly differentiated through their semantics as well as the consequent XML encodings. First, as the stereotype name implies, only one of the values or objects of a Union class can be present at one time. This is implemented by means of the XML <choice> element contained within the Union's <complexType> to specify this restriction. Second, a Union class cannot be derived from another one. In other words, inheritance relationships between Unions are not allowed. Third, the association relationship between two Union classes is also prohibited which means that the values or objects of a Union classes cannot be an instance of a Union class.

Using the UML stereotypes defined by GML and outlined above, a variety of application schemas can be defined as a platform-independent UML data model which can then be automatically implemented as XML-based application schemas. To date, this workflow is well supported by the combined use of the commercial software 'Enterprise Architect' and the open-source software 'ShapeChange' (cf. ShapeChange 2017). 'Enterprise Architect' provides an intuitive graphical user interface that allows developers to interactively design and create UML data models using the UML stereotypes defined by GML. The resulting UML data model can be stored and transmitted as a binary file with the extension '.eap', which is an abbreviation of 'Enterprise Architect Project'. 'ShapeChange' is able to read and parse this binary file and automatically generates the GML application schema according to the XML encoding rules as defined in the ISO 19136 standard. This approach has already been successfully applied in the development of many data schemas e.g. INSPIRE, AAA, and CityGML ADEs (cf. chapter 4).

## 2.5 3D Graphics Visualization

The 3D graphical representation of geospatial data is extremely useful to help people to understand and interpret the real-world information in an intuitive way. Thus, a 3D mapping application is very important for a GIS platform to support the interactive user exploration of the heterogeneous geospatial data like aerial maps, digital terrain models as well as the 3D geometric objects in order to facilitate the application tasks e.g. land management, architectural planning, and urban development (cf. Herman & Reznik 2015, Thompson et al. 2018). However, most of the conventional GIS mapping applications are usually based on 2D and lack 3D capabilities. With the advancements of computer graphics technology and the increasing computational capacities of hardware, 3D geo-visualization has become possible on various devices using different operating systems. In this context, one of the key technologies in building a 3D mapping application is the 3D visualization of the 3D geospatial contents on a 2D screen using modern 3D-enabled Graphics Processing Units

(GPU). Hence, it is important to first get a better understanding of the basic principles of the 3D graphics rendering process (cf. Figure 10).

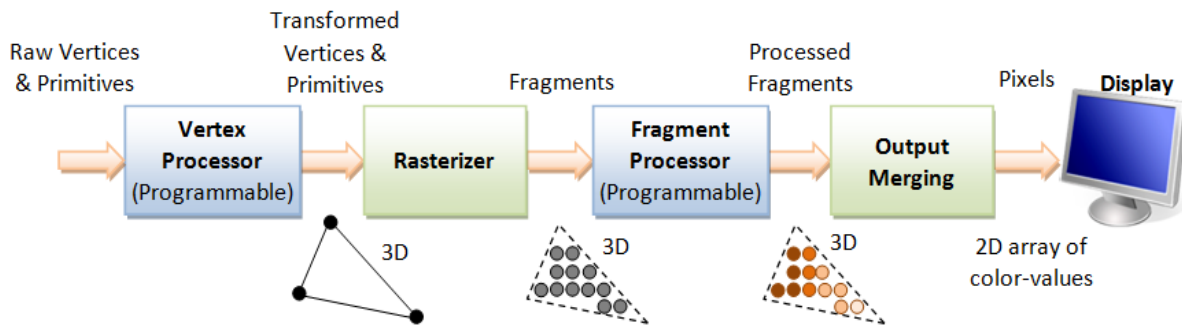


Figure 10: 3D Graphics Rendering Pipeline (Chuan 2012)

In the fields of computer graphics and geospatial information, the well-known *Boundary Representation* (B-Rep) plays a central role for representing polyhedral 3D objects (cf. Foley et al. 1995, ISO 19107:2003). In computer graphics, a B-Rep 3D object is typically composed of a set of surfaces each of which is represented using a number of small polygons in the form like triangles or quads (Parisi 2012). The vertices of these polygons are topologically connected to construct the shape of the 3D object and each vertex has 3D coordinate values ( $x, y, z$ ) expressed in floating point numbers based on a local coordinate system in a 3D Euclidean space. In order to add appearance information to the surfaces, each vertex can be attributed with the additional parameter values like RGB color, materials, as well as textures images which can be draped onto the surfaces by referencing each vertex to the respective 2D texture coordinate ( $s, t$ ) in the image. In addition, each vertex can also have an attached normal vector called vertex-normal which can be calculated by evaluating the normal vectors of the associated polygon surfaces. The vertex-normal is usually used for determining the inner and outer surface: A normal vector pointing inwards or outwards indicates the back or front face of the surface respectively (Chuan 2012). With this information, each polygon will be rasterized into a set of fragments which can be seen as pixels comprising all visual information in 3D spaces.

While moving the camera, the rendering effect of the observed 3D objects may be dynamically changed during the runtime to provide cinematic visualization effects on the 2D screen. This requires a set of rendering rules on how the rasterized fragments should be illuminated according to the camera position in 3D space. Such rules can be defined using the shader technology which allows to process the individual 3D fragments (Kessenich et al. 2008). Basically, shaders are small programs written in a C-like language and allow developers to dynamically define how the pixels for 3D objects actually get drawn on the screen. A shader mainly consists of two parts namely the vertex shader and the fragment shader. While the vertex shader is responsible for transforming the coordinates of the object into 2D display space, the fragment shader controls the rendered color of each screen pixel displaying the mapped vertices according to the input parameter values i.e. color, texture, lighting, and material.

For the visualization of 3D objects, all these information must be carried in an efficient 3D graphics data exchange format. Nowadays, numerous proprietary and open standards have been developed by many software vendors in the fields of movie industry, computer games,

computer-aided design, and manufacturing. One of the popular standards is COLLADA (Collaborative Design Activity) which is an open standard issued by the non-profit consortium Khronos Group (cf. Barnes & Finch 2008). It is an XML-based interchange format identified with a .dae file extension and has been well supported by a range of software tools like 3ds Max, Adobe Photoshop, and Google SketchUp. A proprietary counterpart of the COLLADA standard is the FBX (Filmbox) which was developed by the company Autodesk. The FBX file can be delivered either as XML file or binary file and the latter allows to yield more sophisticated storage efficiency and loading times compared to COLLADA (cf. McHenry & Bajcsy 2008). Although FBX is a proprietary format with limited access to the binary file content, a comprehensive SDK is available allowing applications to read, process, and write FBX files and has achieved a good interoperability between Digital Content Creation (DCC) applications such as 3ds Max, MotionBuilder, and Mudbox. There are many other 3D graphics standards which were specifically designed and optimized for particular usage fields. For example, the web-based 3D graphics standards allow to efficiently draw 3D objects within modern web browsers.

A well-known web-based 3D graphics standard is Virtual Reality Mark-up Language (VRML) which was initiated by the Web3D Consortium. To date, it has been superseded by its successor standard called X3D which supports coordinate georeferencing of 3D objects and is very suitable for GIS applications (cf. Web3D 2015). Besides, the combined use of the KML and COLLADA standards can also provide similar capabilities of displaying georeferenced 3D objects in a 3D mapping application. However, in earlier years, the direct web-based visualization of these 3D models is almost impossible since the rendering of large 3D models with detailed geometric and appearance contents usually requires very high bandwidth and computing resources which were not supported by the conventional web browsers. To overcome this issue, a common solution was to install additional plugins into web browsers in order to access and make use of the computer graphics API for rendering the 3D objects in a 3D window embedded into web browsers. However, such plugin-approach exposed many security issues due to which the installation of additional plugins is nowadays strictly prohibited by most modern web browsers. As a result, the plugins for viewing X3D, KML, and COLLADA 3D models have been deprecated as well. Against this background, the way of web-based 3D visualization has been revolutionized with the development of the new technologies HTML5 and WebGL.

HTML5 is an Open Standard format and provides a common platform for applications to be developed and used on the web (cf. Faulkner et al. 2017). HTML5 enables the new generation browsers to support multi-threading, which allows to perform parallel execution of different tasks within one web page. WebGL is an implementation of the OpenGL API for the web and extends the HTML5 canvas element to utilize the computer graphics card to provide hardware accelerated 3D functionality for the web browsers running on different devices such as smart phones, desktop and tablet computers (cf. Jackson 2014). Applying such an approach, 3D capabilities can be fully realized in all major web browsers running on all major operating systems without needing additional plug-ins or extensions. However, WebGL is a low-level API and very complex in nature. In order to avoid low-level programming, a framework wrapping WebGL would be very helpful for developers to quickly and easily handle WebGL content in a small piece of code. Nowadays, a number of WebGL-based frameworks are available such as *X3DOM*, *Three.js* and *Scene.js*, which come with extensive libraries for

building web-based 3D mapping applications to visualize 3D graphic models directly (cf. Krämer & Gutbell 2015).

Since the aforementioned 3D graphics standards like COLLADA and X3D were not designed for web-based 3D visualization, it is difficult for web browsers to yield the optimal processing performance when viewing these 3D models using HTML5 and WebGL technologies. Considering this issue, a new 3D graphics format called glTF (GL Transmission Format) has been issued by the Khronos Group. It is a royalty-free specification for the interoperable use across the industry. Compared to the other 3D graphics standards, glTF has a JSON-based file structure and can also be formatted as binary file to minimize the file size allowing for fast transmission over the Internet and efficient processing time at the client side. Besides, it supports all relevant features of a 3D graphics format for carrying the 3D content information like 3D positions, materials, animations, skins, cameras, and lights, which can be easily converted from other 3D graphics formats. Furthermore, an outstanding feature of glTF is the support of an extension mechanism which allows application vendors to add new properties and parameter semantics to the glTF models with respect to the glTF specification (cf. Bhatia et al. 2017).

For the sake of clarity, a rough comparison of the aforementioned 3D graphics formats is summarized in Table 1 based on their latest versions (cf. Burggraf 2015, Barnes & Finch 2008, Web3D 2015, Bhatia et al. 2017, Autodesk 2014). The chosen comparison criteria are openness, the support of geometry, georeferencing, appearance, animation, and thematic attributes.

Table 1: Comparison of the key features between different 3D visualization models

	COLLADA	FBX	KML	X3D	glTF
openness	++	0	++	++	++
geometry	+	+	++	++	+
georeferencing	0	0	+	+	+
appearance	++	++	+	++	++
animation	++	++	+	++	++
thematic attributes	+	+	+	+	+

Legend: 0 = not supported, + = limited, ++ = comprehensive

The first comparison criterion is ‘openness’, which reflects the degree of standardization of a 3D graphics format for public use. This criterion is well fulfilled by COLLADA, KML, and X3D, as they are open standards issued by the standards organizations like OGC and Khronos Group. FBX is however a proprietary file format and it has no open documentation about how



to parse and interpret the files. The second aspect for the comparison is the capability of representing geometry information. Both KML and X3D allow to represent the primitive geometries e.g. point, curve, and polygon, while COLLADA, FBX, and glTF lack the support for representing single point. Regarding the support of georeferencing, both COLLADA and FBX are not able to represent georeferenced 3D objects. Although KML, glTF, and X3D allows mapping 3D objects from scene coordinate system to a world coordinate system, they all have limitations. For example, both KML and glTF solely support the WGS84 geographic reference system, and X3D only supports two additional types, namely Universal Transverse Mercator (UTM) and WGS84 geocentric coordinate reference systems. The storage and description of appearances e.g. textures and animations e.g. skeletal animations are well supported by these formats except KML, which solely supports simple color styles and time-based animations. Strictly speaking, all these graphics formats allow to encode additional data like thematic attributes for 3D objects. For instance, KML offers a tag element called 'description', which can be used for associating an 3D object with custom data encoded as plain text or HTML content. However, the specifications of all these formats do not standardise or specify the way how the encoded thematic attributes data can be parsed or validated by applications.

## 2.6 Digital Virtual Globes

A digital virtual globe is a three-dimensional virtual environment for representing the Earth surface in a realistic way (cf. Cozzi & Ring 2011, Grossner et al. 2008). The fundamental component of a virtual globe is a 3D globe which allows users to navigate and explore the Earth map by panning, moving, tilting, and rotating the camera perspective using a mouse or touchscreen. Typically, a virtual globe is capable of representing many different geographic features as 3D geometries or surface meshes. In addition, different types of base maps such as satellite images and digital terrain models can also be draped onto the Earth surface to facilitate the recognition of viewing locations and orientations which are helpful for the urban planning process (cf. Hu et al. 2010). Moreover, some utility functionalities such as geolocalization, selection of different base layers, and switching between different viewing modes are also supported by most virtual globes.

The first widely published virtual globe is Google Earth developed by Keyhole and later bought by Google Inc. in the year 2004. It was first released under the name 'Google Earth' in 2005 and is nowadays still one of the most popular virtual globes for free use. Google Earth fully supports the display of satellite images, aerial photography, topographic maps as well as KML files combined with COLLADA models for supporting the display of textured 3D objects. It can also be operated on various mobile and desktop platforms using different computer graphics API like OpenGL or direct3D. For the web visualization, Google Earth came with a browser plugin called Google Earth Plugin along with a comprehensive API allowing developers to build their own 3D mapping applications on top of the Google Earth globe. However, due to the browser plugin prohibition, the APIs have been deprecated since December 2015. In order to achieve an alternative solution, a number of WebGL-based open-source virtual globes have been developed such as WebGL Earth (cf. WebGL Earth 2012), OpenWebGlobe (cf. Loesch et al. 2012), and CesiumJS (cf. CesiumJS 2019). Nowadays, due to its extensive support of the efficient presentation of 3D geospatial data on web browsers,

CesiumJS has been evaluated as the most powerful virtual globe solution, which has been used by a large number of applications world-wide (cf. Krämer & Gutbell 2015, CesiumJS 2019). The main features as well as the software structure of CesiumJS is roughly illustrated in Figure 11.

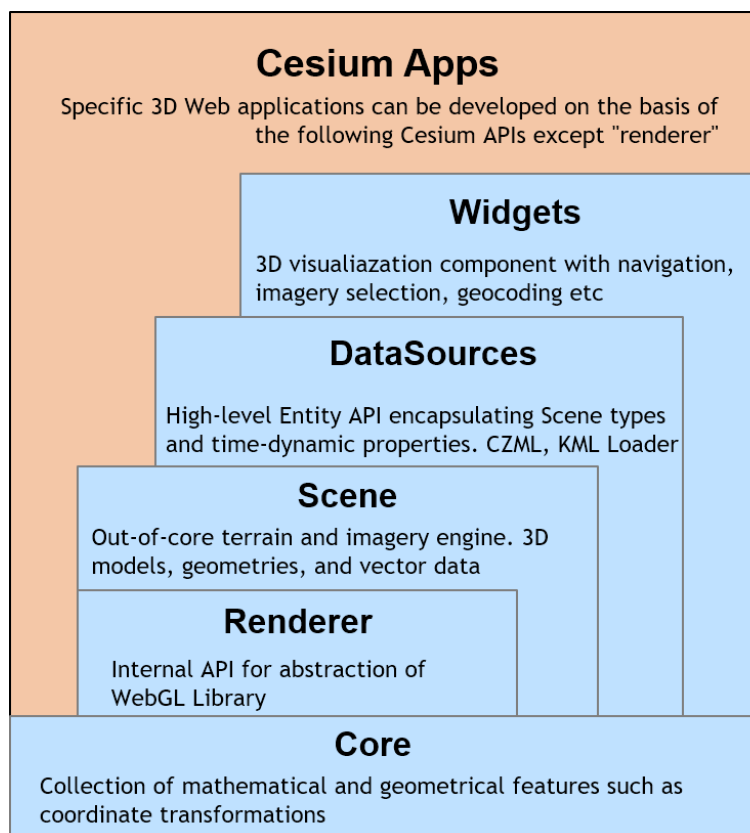


Figure 11: API architecture of the CesiumJS framework library (Cozzi 2015)

As shown in Figure 11, the CesiumJS APIs are organized in a hierarchical structure with different layers with regards to the specific functionalities and abstraction levels. The *Core* is the lowest layer in Cesium and includes a number of static utility functions such as mathematical conversions, coordinate transformations, and projections etc. The *Renderer* is a thin abstraction over the WebGL library and comprises a set of GLSL functions to provide shader programs, textures and buffers which are not exposed and just for internal use. The layer *Scene* provides the high-level globe and map constructs such as 3D globe or map, handling layer images from multiple sources like Web Map Service (WMS) and Tile Map Service (TMS), creation of geometries and materials, camera control and animation. The layer *DataSources* offers a high-level Entity API allowing to load geospatial data from different types of data sources according to different formats such as CZML, KML, and GeoJSON. all of which can be handled using a common data source interface. The *Widgets* is the top-most layer providing a so-called *Cesium Viewer* (cf. Figure 12) which is a composite GUI widget shipped with the Cesium API and provides overall functionalities of a 3D globe such as camera control, rendering geometries and materials, animation etc. In addition, the *Cesium Viewer* contains a number of helpful widgets and plugins that provide functionalities like querying of geocoding service, switching between different viewing modes (2D, 2.5D, and 3D view), and that manage imagery and terrain layers easily. With the help of the Cesium API, it is also possible to add additional widgets as 'plugins' to the *Cesium Viewer* to add

further functionalities and features by third parties. Thus, the *Cesium Viewer* can serve as a good starting point for developers to easily extend it to build customized 3D mapping applications.

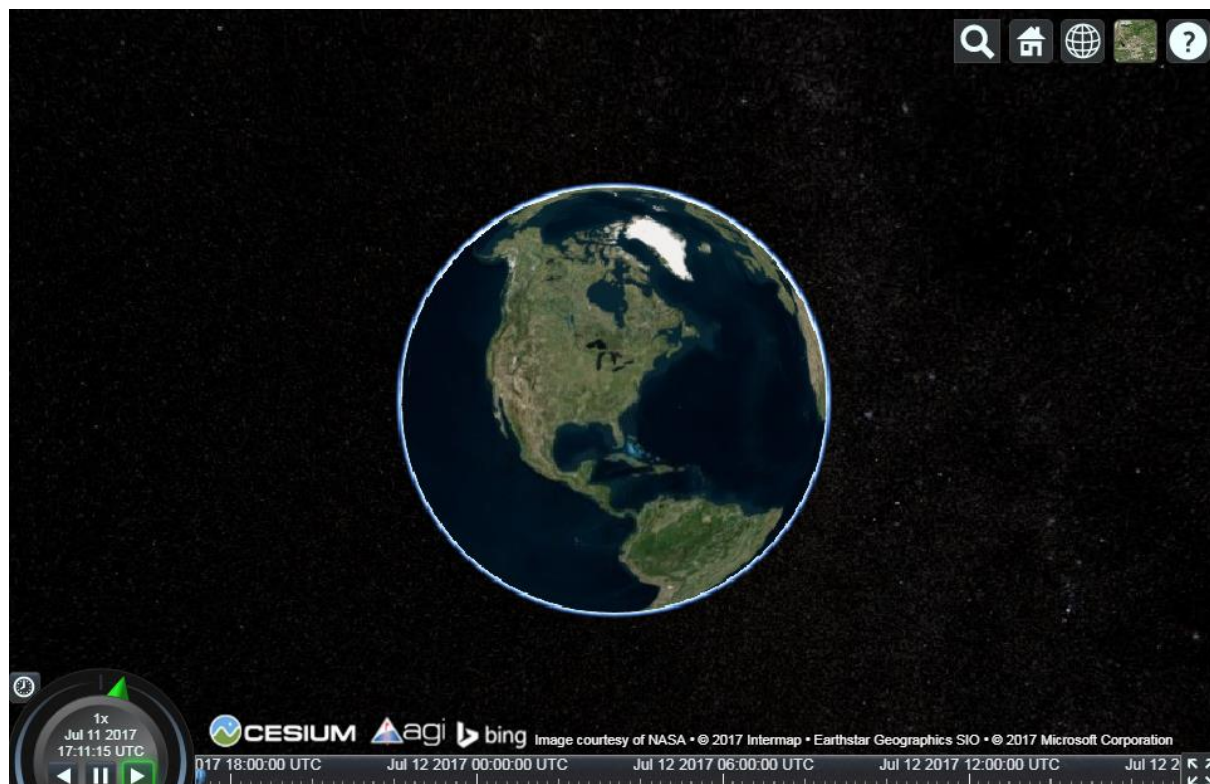


Figure 12: Graphical user interface of the *Cesium Viewer*



## Chapter 3 Management of Semantic 3D City Models

As complex geospatial data models, semantic 3D city models are complex mappings and inventories of the urban environment including not only the man-made objects like buildings and infrastructures but also the natural features like vegetation, water bodies, terrains etc. All these city objects along with their sub-parts are completely represented with homogeneous data quality regarding the geometric, topological, visual as well as thematic properties. With such an integrative data model, the semantic 3D city model can serve as an information carrier allowing for the exchange of geospatial data between different application domains. Thus, the semantic 3D city model can be used as a common ontology to describe the cities and allows GIS companies to develop advanced applications that can fully exploit the spatio-semantic urban information being required or useful for a wide range of applications like urban planning, environmental and training simulations, disaster management, and energy assessment.

Due to the highly complex data structure, the management of semantic 3D city models in GIS applications exposes many challenges. First, a universal information model is required to provide a common definition of the model entities, attributes, and relations along with a standardized exchange format that can be interpreted by applications using a common interface. Second, in order to ensure the interoperable access to the model information, the database technology is considered to be the most powerful means for supporting the persistent and efficient data storage of large 3D city models. To realize this, the database schema i.e. the relational database schema must be carefully designed to create a compact database structure allowing to efficiently perform the database operations like reading, writing, querying, and updating of the thematic as well as the spatial data by making full use of the spatial capabilities of the spatial database systems. Third, a complete software toolkit is needed to provide the relevant functionalities supporting the work chain, starting from the reading, processing, and writing of the data contents in the database, via the conversion to different model representations, up to the high-performance data visualization and exploration within a 3D mapping application.

This chapter aims to give a comprehensive review of the existing developed approaches for answering the above-mentioned three challenges and, hence, has been structured into three parts accordingly. The first section gives a brief introduction of the international standard CityGML which is an important approach for modelling, maintaining, and exchanging semantic 3D city models. Actually, all the research and implementation work carried out in the course of the thesis are based on this city model standard for answering the research questions outlined in the first chapter of the thesis. In the second section, emphasis is placed on the essential aspects of realizing the efficient management using spatially-enhanced database system. The relevant relational database modelling approach is discussed in detail to provide the foundation for designing a highly efficient and compact relational database structure according to the CityGML standard which is an object-oriented data model. The last section presents an open-source software toolkit called 3D City Database (3DCityDB) which is an extensive software solution based on spatial relational databases for managing CityGML-compliant semantic 3D city models. It additionally comprises a set of software tools which can be used combined with other GIS and ETL software for building sophisticated applications to accomplish various domain-specific analysis and simulation

tasks. All these software tools will be presented and explained in detail with respect to the technical implementations and the conceptual ideas of software design.

## 3.1 CityGML

### 3.1.1 Overview

The City Geography Markup Language (CityGML) is an open data model and international standard issued by the Open Geospatial Consortium (OGC). It has been originally developed by the Special Interest Group 3D (SIG3D), which is an international working group consisting of many companies and academic organizations who aim to create an open and interoperable 3D spatial data infrastructure based on the existing international standards. The first official version of CityGML was released in the year 2008 and its latest stable version is 2.0.0 having been published in March 2012. Under the leadership of the chair of Geoinformatics at Technical University of Munich (TUM), the development of the next major version of CityGML was initiated in 2014 which will include many changes to the current model structures, concepts as well as a number of additional semantic modules regarding the change requests submitted by the CityGML users (cf. Löwner et al. 2014).

Basically, CityGML defines a rich feature catalogue and platform-independent UML data model for the most relevant 3D topographic features like buildings, bridges, waters, vegetation etc. All these feature classes including their geometrical, topological, visual, and semantic properties are modelled in a modular fashion (cf. Figure 13) to provide a complete 3D city model and a common definition of the relevant urban objects within one framework. Besides, the data model has been mapped onto an XML-based application schema using OGC's Geography Markup Language (GML3) for data exchange between different GML-aware applications as well as the automatic data validation against an XML schema definition file using software programs. Thus, CityGML can be employed as an integrative data model with interoperable exchange format to facilitate those GIS applications that rely on a complete semantic 3D city model.

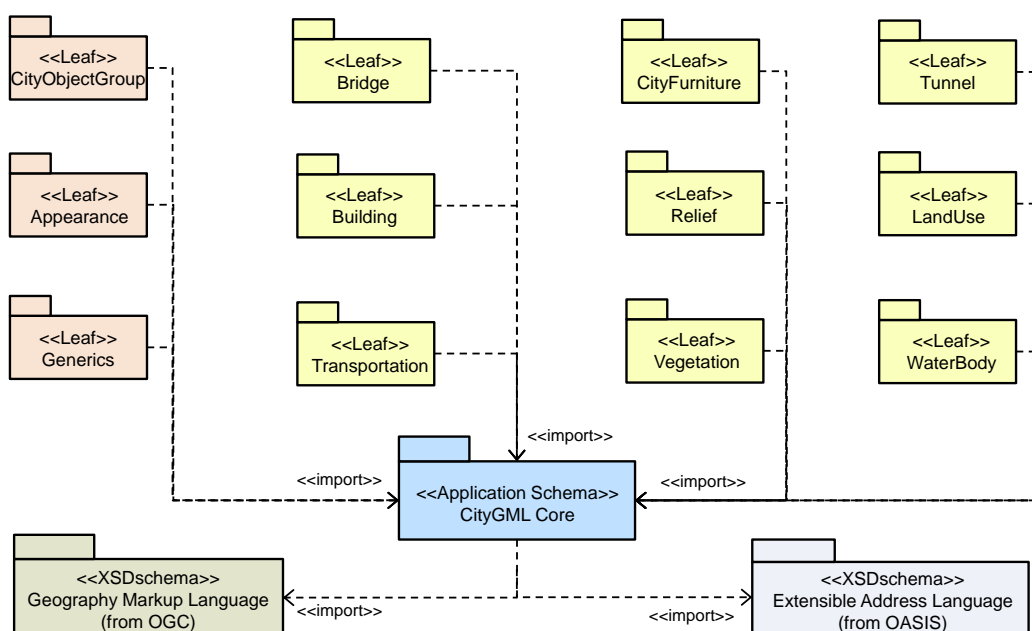


Figure 13: Overview of the CityGML modules (cf. Gröger et al. 2012)

In CityGML, all classes and data types are grouped into individual modules according to the thematically differentiated classes representing the relevant city objects. The conceptual relationships between different modules are described using a UML package diagram (cf. Figure 13) and implemented as a set of packages which are assigned with globally unique namespaces for the corresponding XML schema definition files. In addition, the dependency between different modules is represented using a dashed arrow tagged with the stereotype <<import>> for denoting which schema must be referenced by another one. For example, the *CityGML Core* module defining the basic concepts and components of CityGML is a mandatory package that must always be imported into the XML packages of the other CityGML's modules including *Building*, *Bridge*, *Transportation*, *CityObjectGroup*, *Appearance*, *Generic*, *CityFurniture*, *Relief*, *Vegetation*, *Tunnel*, *Landuse*, and *WaterBody*, which are marked with yellow color in Figure 13. In addition, since CityGML is based on the GML model, the *CityGML Core* module has a dependency of the GML schema which must always be imported by the CityGML schemas. Another package referenced by the *CityGML Core module* is the OASIS' Extensible Address Language (xAL) which maps the address formats of the different countries into a unified XML schema which has been usually used for encoding the address information of a building object in a standard XML structure. In the future version of CityGML, additional modules will be added including e.g. the *UtilityNetwork* module for describing the underground network infrastructure and the *Dynamizer* module which allows handling dynamic properties such as real-time sensor data acquired over time (cf. Becker et al. 2013, Kutzner & Kolbe 2016, Chaturvedi & Kolbe 2016).

### 3.1.2 Main Features of CityGML

According to the CityGML specification (cf. Gröger et al. 2012), the CityGML modules are modelled with numerous outstanding features some of which are summarized in the following.

#### *Level of Details (LoD) Concept*

CityGML defines a level of detail (LOD) concept supporting variations of an individual city object regarding its spatial and semantic resolutions in the conceptual meanings. Most of the CityGML features including the digital terrain model can be simultaneously represented in five discrete resolutions (0 - 4) ranging from coarse models (LOD0) to geometrically and semantically fine-grained structures (LOD4).



Figure 14: The LOD concept defined by CityGML (Biljecki et al. 2016)

For example (cf. Figure 14), a CityGML building object can be simply represented using its LoD0 footprint or a 3D solid (LoD1) which can be calculated by extruding the 2D footprint according to the building height to construct a simple geometrical 3D representation. Starting



from LoD2, buildings can be semantically enriched by classifying their outer shell into different types of thematic surfaces such as wall surfaces, roof surfaces, and ground surfaces. In LoD3, doors, window, as well as extensions of buildings like balconies and stairs can also be thematically modelled for refining the semantic of buildings. Moreover, LoD4 completes a LOD3 model by adding interior constructions within a building which can be further composed of rooms, interior installations like doors, stairs, and furnitures. The same LOD modelling approach also applies to other CityGML feature types like bridges and tunnels. The main advantage of such LOD concept is that different degrees of scale referring to the same city object are available to the application users allowing them to flexibly choose the suitable resolution according to the different application requirements. For example, the LOD0 model is very efficient for the visualization on a 2D or 3D map and the higher LOD models (LOD  $\geq 2$ ) provides the complete information allowing for performing the analysis like i.e. the building volume and floor area calculation, energy demand estimation, and solar potential analysis.

### Grouping Concept

The aggregation relationships between city objects can be properly represented in CityGML by using the class *CityObjectGroup* which belongs to the *CityObjectGroup* module. It is a subclass of the *\_CityObject* class and allows to reference multiple city objects to create a group. In addition, the instances of *CityObjectGroup* can also be aggregated to build a recursive aggregation hierarchy with arbitrary levels (cf. Figure 15). Moreover, the *CityObjectGroup* class has a spatial attribute called ‘*geometry*’ which can use a polygon geometry to describe the spatial extent of a *CityObjectGroup*. This allows to ensure the spatial coherence which means that all group members must be completely inside the bounding area of their parent group. For example, a number of buildings might be grouped according to the administrative districts which can be aggregated again to a city. With this aggregation approach, it is possible to rapidly enumerate and visit all members of a *CityObjectGroup* at any aggregation level. This is especially helpful for applications to automatically perform statistic functions like count, sum, and average the values of a set of numeric attributes for all member objects belonging to the same *CityObjectGroup*. For example, the energy demand values from all buildings within a district can be easily aggregated and the resulting values can be added as new attributes to the parent *CityObjectGroup* and propagated to the root *CityObjectGroup*.

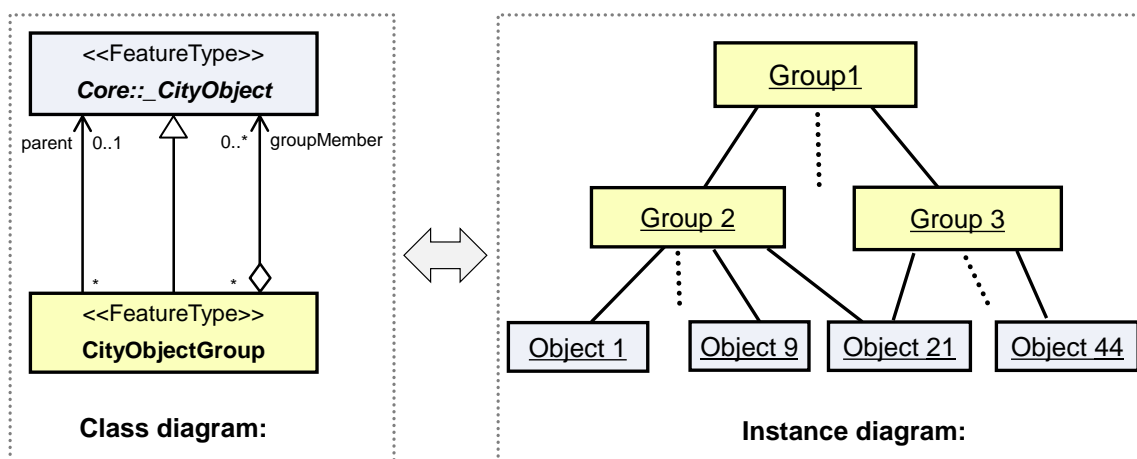


Figure 15: Example of realizing an aggregation hierarchy using CityGML's grouping concept





Foley et al. 1995). For example, a solid is a closed geometry which is bounded by one exterior and zero or more interior shells represented as surface geometries. In addition, each surface geometry can also be attributed with an explicit orientation information using the geometry type *OrientableSurface* which allows to distinguish its front and back sides. This B-Rep approach is very appropriate for 3D city modelling since it allows to explicitly represent the shape of a 3D model object using explicit elements which can be easily attached with additional information i.e. appearances using their IDs. However, it would result in redundant vertices and edges at the boundaries which are not efficient for the data storage and 3D visualization especially in the case of using the geometry type *TIN* for handling massive digital terrain models (cf. Kumar et al. 2016).

In order to represent topological relationships between geometries, CityGML uses the XLink concept originated from the GML specification. Each geometry object that should be shared by different composite geometries is assigned a unique identifier which can be referenced by a remote geometry element located in the same document. In addition, the spatio-semantic coherence in the CityGML building model can also be ensured using the XLink mechanism. For example (cf Figure 17), most of the LoD2 CityGML building models produced today both semantically describe the wall, roof and ground surfaces and additionally provide a solid geometry for the geometric representation of the outer shell and 3D shape. The semantic parts are usually used to query and analysis the separate parts of the building, whereas the solid geometry represents the whole body and is useful for geometric calculations such as deriving the building's volume and surface area. Both ways of describing the building are complementary and not mutually exclusive and therefore provide a very flexible modelling structure ranging from simple geometric models to semantically rich models. Since the geometry of a wall or roof surface can be identical to a part of the solid geometry, the reuse the surfaces geometry can be realized using an XLink.

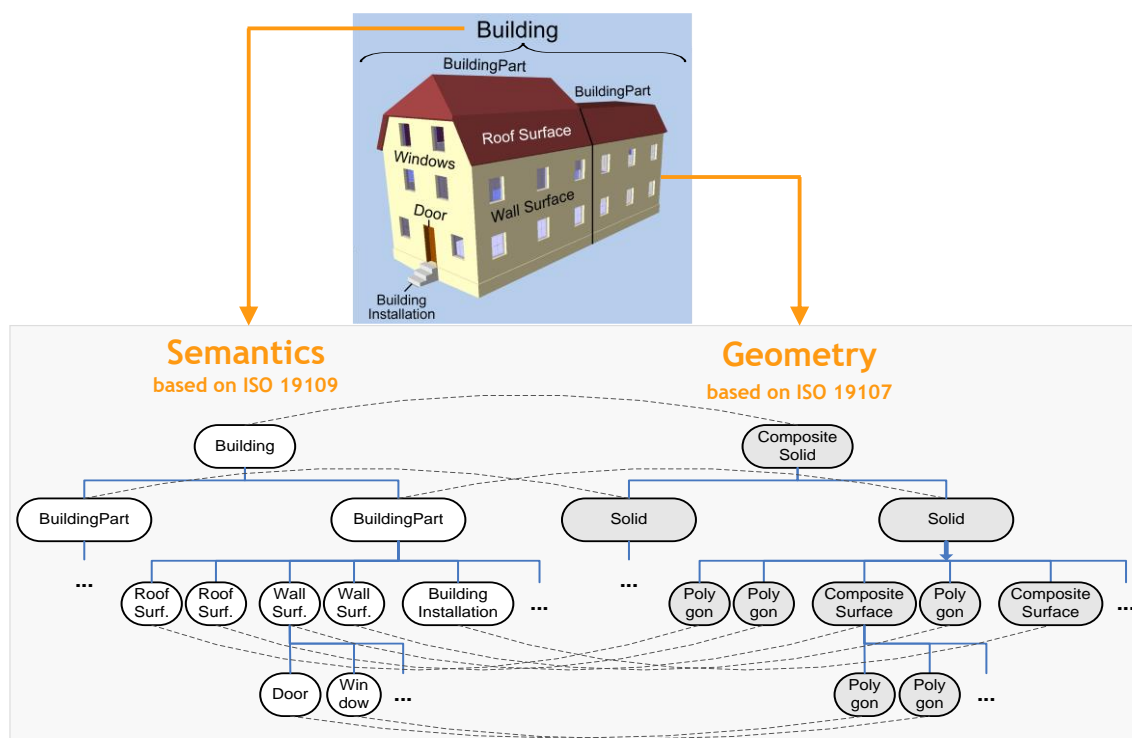


Figure 17: Coherence of semantics and geometry in CityGML (Stadler & Kolbe 2007)

### *External references for all city model object*

In CityGML, each city object can be linked with an arbitrary number of external information systems e.g. cadastral databases, telecommunication databases, and facility management systems which allow for supplying additional information referring to the same building object. This is realized using the CityGML concept ‘ExternalReference’ which is implemented as a data type called *ExternalReference* associated with the *\_CityObject* class. Each city object allows to have multiple external references each of which consists of an URL of an external system and an object name or ID being a unique identifier within the respective system. The combination of these path information builds a unique address of the external data contents of a city object which can be easily accessed over the Internet. For example (cf. Figure 18), if a building is linked with three external information systems, three corresponding URIs linking with the external object contents can be generated. These URIs can be displayed in a tabular form when the building object has been selected in a GIS mapping application where each URI can be decorated as a hyperlink allowing users to simply click on it to launch the data query against the respective information system. As a result, the returned data contents will be displayed in a new dialog window.

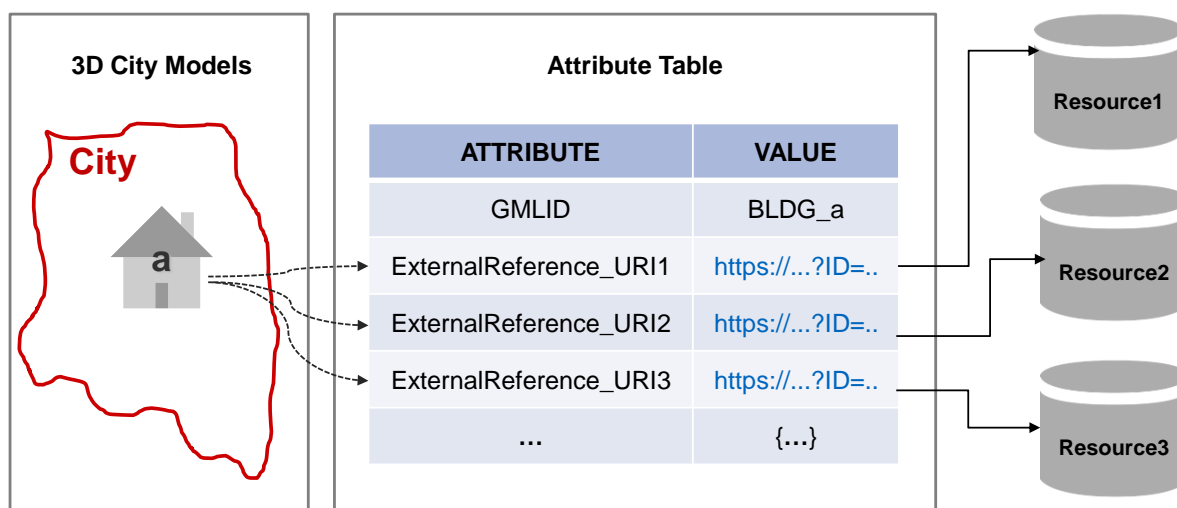


Figure 18: Example of using CityGML External References for linking with remote data repositories

### *Support of extension mechanisms*

In practical applications, an essential issue often encountered while using CityGML is that many additional feature classes or extra attribute types are required which are not defined in the standard CityGML modules. To overcome this issue, CityGML provides a simple extension approach defined in the *Generics* module, based on which a *CityObject* can be enriched with an arbitrary number of additional generic attributes using the CityGML data type *\_genericAttribute* or modelled as a generic city object using the CityGML class *GenericCityObject* without needing to alter the CityGML application schemas. Supported data types of a generic attribute include the primitive types such as string, integer, floating-point numbers, and date. In addition, multiple generic attributes with different data types can also be aggregated to form a new generic attribute called *genericAttributeSet* which is a subtype of *\_genericAttribute*. In this way, a complex attribute with arbitrary aggregation levels can be created to augment the CityGML objects like building, road, tunnel, etc. with simple or complex-structured information. The *GenericCityObject* is a subclass of the

*CityObject* class and is typically used for representing any feature types that are not explicitly defined in CityGML. Like with the other CityGML top-level feature classes, each *GenericCityObject* is attributed with the properties like *class*, *function*, *usage* as well as spatial properties with different geometry types for representing the *GenericCityObject* at different detail level.

Another extension solution is called Application Domain Extension (ADE) which provides a more systematic solution for extending the existing CityGML models. Each ADE can be seen as a separate module which references the dependent CityGML standard modules (cf. Figure 19). This way, it offers a high degree of flexibility to the developers and allows them to design a complex data model. In addition, due to the modular model structure, it is also very simple to merge the developed ADEs into the CityGML framework to become new standard CityGML modules. Moreover, since each ADE is a data module and can have an XML application schema, this allows a software program to automatically validate the ADE instance documents during runtime to guarantee the data correctness and consistency which is typically not supported by utilization of the CityGML's *Generic* module. Thus, the ADE approach is considered the most reliable approach compared to the *Generic* approach in practical applications. However, since an ADE introduces a completely new application module, the complexity of the data models as well as the related software implementations are substantially increased. More detailed discussions about the handling of CityGML ADEs are elaborated in the next chapter.

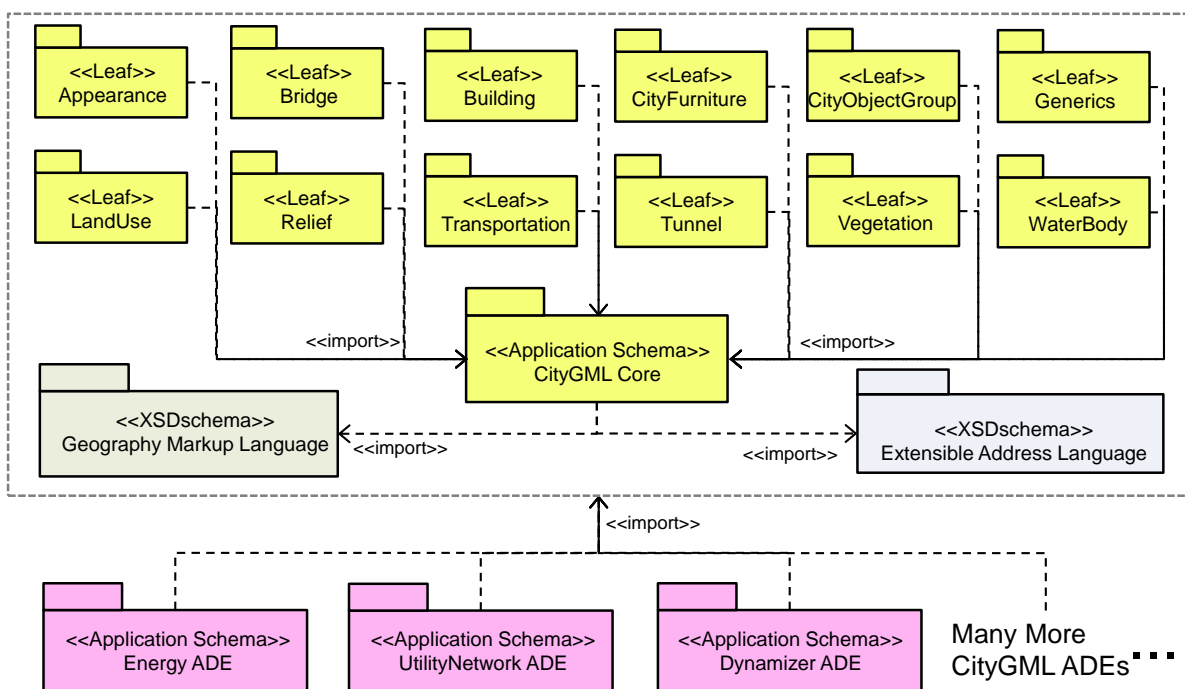


Figure 19: Relationship between CityGML and ADE modules

### 3.2 Management of CityGML using SRDBMS

For the efficient storage and management of CityGML datasets, the spatially-enhanced relational database system is considered to be a powerful means which also allows for interoperable access from application programs or ETL software. The conceptual solution of designing a relational database schema for handling data with object-oriented structures can

be traced back to the old problem of mapping object-oriented model onto relational data model. Both data models are categorized under different paradigms but can represent the same data models semantically (cf. Kutzner 2017). An analysis of the existing relational database systems indicated that a more compact database schema is much more efficient for querying and processing of large and complex-structured data to achieve a good performance while interacting with the database (cf. Stadler et al. 2009). To reach this goal, the database schema for storing CityGML data contents shall be resulted from a careful assessment process by mapping the CityGML data models onto a compact relational database structure with respect to the database complexity, operating performance, and semantic interoperability. This will allow the database to serve as an efficient as well as consistent data layer for transforming i.e. building information between different domains. For example, IFC data could be imported into a CityGML database by spreading them over the respective tables such that CityGML datasets can be easily derived from the database and processed by the CityGML-aware applications.

The approaches of mapping object-oriented models onto relational database models have been extensively studied and discussed in many literature of the past 30 years. For instance, (Golobisky & Vecchiotti 2005) summarized the fundamental concepts for deriving relational database schemas using different mapping rules according to the source UML class structures. A comprehensive discussion on the comparison between different mapping rules has been given by (Keller 1997) to help the database designers to choose the optimal mapping approaches according to the various application requirements. Moreover, Kolbe et al. 2017 introduced a number of advanced mapping rules for designing a compact and optimized relational database schema which allows the spatially-enhanced relational database management system to efficiently deal with large CityGML datasets with complex-structured thematic and spatial properties.

In this chapter, the relevant mapping rules proposed in the above-mentioned literature for designing a CityGML database schema are briefly reviewed and presented. They are grouped into two categories which are illustrated in separate subsections. The first group refers to the standard mapping rules which are originated from the earlier literature and provides the fundamental concepts for the relational database modelling of object-oriented data models. The second category corresponds to the advanced mapping rules that are specifically designed for dealing with complex spatial data model. The main objective of this review is to identify the logics of these mapping rules for paving the way of developing a computer program that allows to derive a compact database schema automatically. The related details are elaborated in the next chapter.

### **3.2.1 Standard Approach for Mapping of OO-Models onto Relational Models**

Depending on the relationships between classes, a set of class patterns have been identified to define the basic mapping rules for the derivation of a relational database model. These patterns are summarized as follows.

- Mapping a single class or complex data type
- Mapping inheritance (Generalization/Specialization) relationship
- Mapping N:0..1 association

- Mapping 1-N association, aggregation and composition
- Mapping M:N associations

For some of these patterns, there may exist more than one mapping rule, which typically differ in terms of query performance and storage efficiency.

### *Mapping a single class*

Normally, a class shall be mapped onto one table (cf. Figure 20) where each row should represent an instanced object of the class. Thus, the mapped table shall have at least one primary key column which can be named as “ID” and defined with a numeric data type for storing the object identifier which must be unique within the table. Additional columns can also be added to the mapped table for storing the scalar attribute values of the respective objects or serving as foreign keys linking with other tables where the non-scalar attributes are stored.

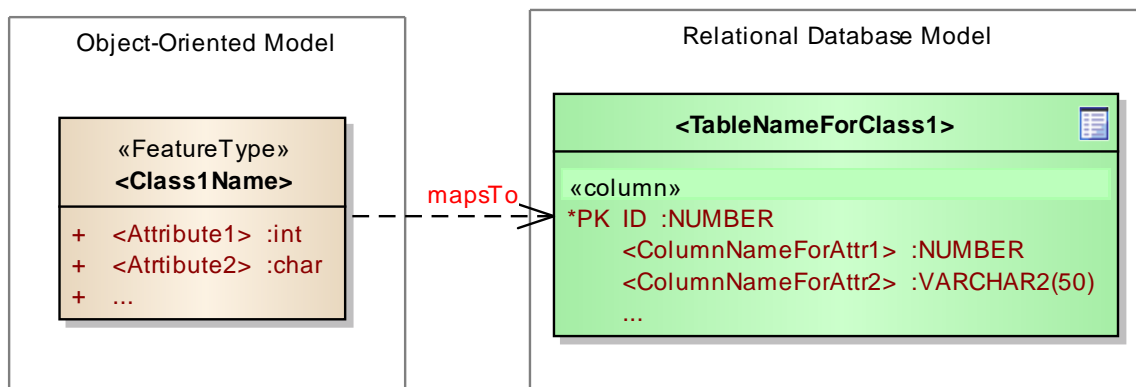


Figure 20: Mapping a Class onto a table

The table name should be somehow identical with the class name, which allows human operators to quickly recognize which table is mapped from which class. However, the two names can sometimes differ due to two reasons. First, a class name could be started with an underscore character which is conventionally used for denoting an abstract class in an object-oriented data model. However, most database systems prohibit that the table name shall not be started with a special character like underscore or digit. In this case, the class name with omitted leading underscore can be used as the table name. Second, in case that the class name contains over 30 characters, the table name must be shortened, because ‘30’ is the maximum allowable length for a table name in some relational database systems like Oracle. Besides, if the tables in an Oracle database have been versioned using the Workspace Manager, the name length of each table will be strictly limited to 25 (cf. Oracle 2017).

In order to achieve a good shortened table name, a proper shortening-rule needs to be carefully designed, because it is not possible to simply truncate the original class name from its start to the position of the limited length. For example, if a class is named as a compound word like ‘*IndustrialBuildingConstructionComponent*’ which is based on the camel case structure being frequently used for naming model classes, the truncated result will be ‘*IndustrialBuildingConstru*’ which is obviously not able to reflect the actual meaning of the original class name. Instead, a good approach is to first split this long word into several simple words like ‘*Industrial*’, ‘*Building*’, ‘*Construction*’, and ‘*Component*’. In the next step, according to the total length limit of the target table name, each of these words shall be



truncated or abbreviated and then combined with each other using underscores. A possible result could be ‘*Indust\_Build\_Constr\_Compo*’ which is a sophisticated name for the mapped class table and can also be easily generated programmatically according to this conceptual shortening-rule.

### Mapping inheritance relationship

One solution for the mapping of an inheritance relationship between two classes is to use a foreign key constraint to link the subclass table with the respective super class table by joining their primary keys, where the primary key in the child class table is at the same time a foreign key of the parent class table (cf. Figure 21). This allows to explicitly represent the inheritance relationship in the relational database model and conforms best to the object-oriented concepts. This approach is also able to ensure the data integrity and consistency using the database’s CASCADE mechanism. For example, if an instance of the child class has a record in its mapped table, a new record with the same ID must be added into the table representing the super class to store the values of those attribute that are inherited from the super class. Once a record has been removed from the parent table, the corresponding record in the child table can be automatically deleted by the database using its ON DELETE CASCADE action.

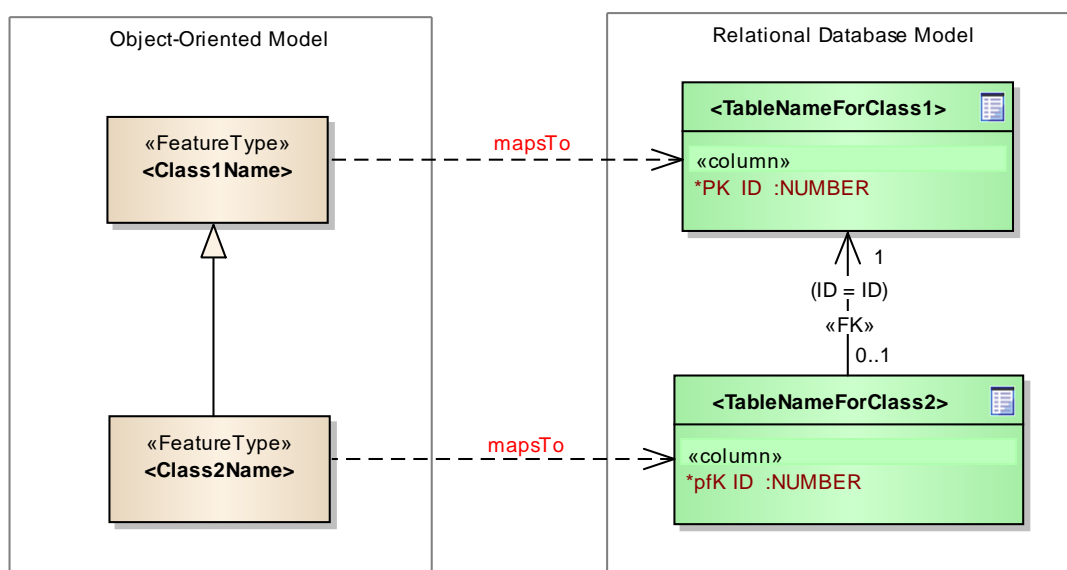


Figure 21: Mapping each class of an inheritance hierarchy onto a separate database table

Another approach for handling inheritance relationship is to map both the child and super class onto one table (cf. Figure 22). Since all instance objects of these two classes are stored in the same table, an additional column named like *OBJECTCLASS\_ID* shall be introduced which can be used to determine the class affiliation of each record in the table. Compared to the upper-mentioned mapping approach, this one is considerably more efficient for database query procedures, since additional database join operation for linking two tables can be avoided. However, in case that the two classes have very different attributes, this mapping approach will result in a lower data storage efficiency, because many cells in the table will be filled with NULL which also consume space on the hard disk. There are many additional design trade-off between these two approaches which are comprehensively summarized by (Ambler 2000) and (Awang & Labadu 2012).

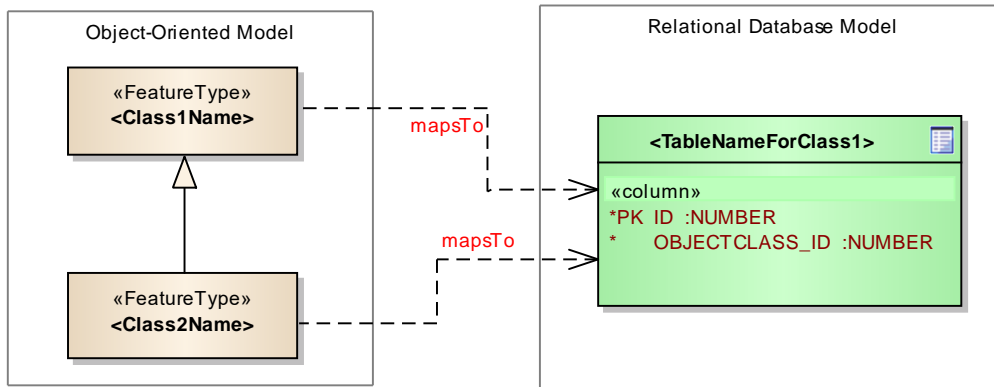


Figure 22: Mapping two classes with inheritance relationship onto a single database table

**Mapping N: 0..1 Association**

There are three typical variants when dealing with the N:0..1 association relationship. The first case is that two associated classes are mapped onto two separate tables (cf. Figure 23), where a foreign key column shall be added into the table of the source class and at the same time references to the primary key column of the target class table.

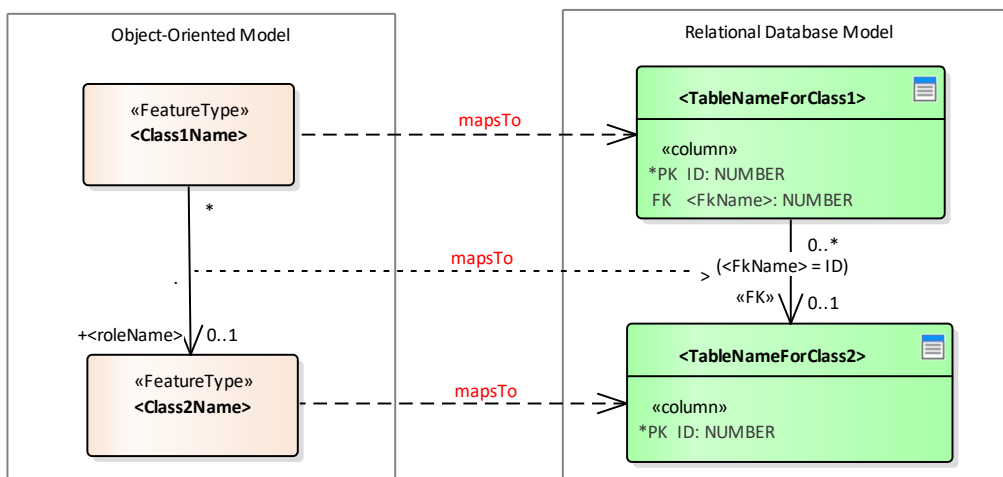


Figure 23: Mapping N:0..1 relationship between two different classes (variant 1)

In case that two classes are mapped onto one table (cf. Figure 24), a foreign key column shall be added which at the same time references to the primary key column of the same table.

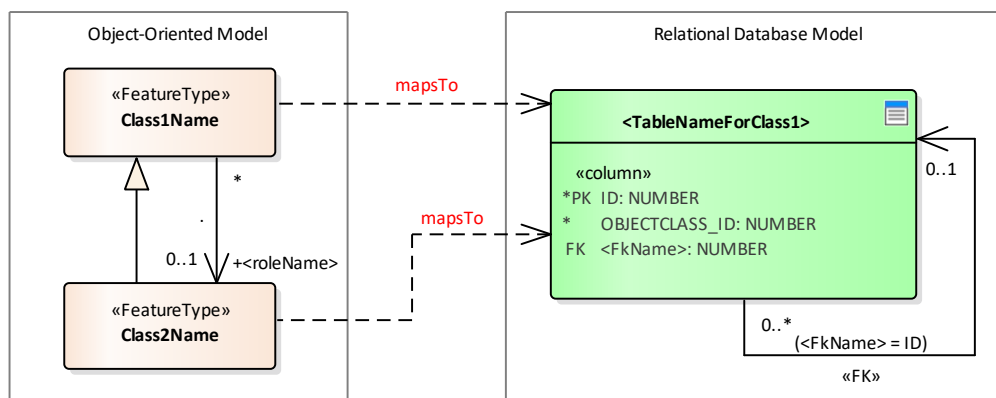


Figure 24: Mapping N:0..1 relationship between two classes with a shared table (variant 2)



The third case is that a class is associated with itself and mapped onto one table which shall hold the foreign key column and its referenced primary key column (cf. Figure 25).

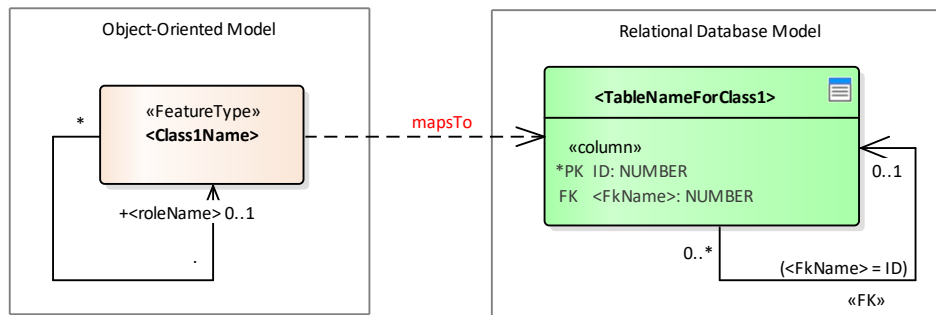


Figure 25: Mapping N:0..1 relationship of the same class (variant 3)

**Mapping 1:N Association (Aggregation/Composition)**

The mapping of 1:N association can also be realized by using a foreign key column which shall be added into the target class table and points to the primary key column of the source class table. The same approach can also be applied for the aggregation and composition relationships since both are actually the special cases of the 1:N association with specific semantics (cf. Figure 26).

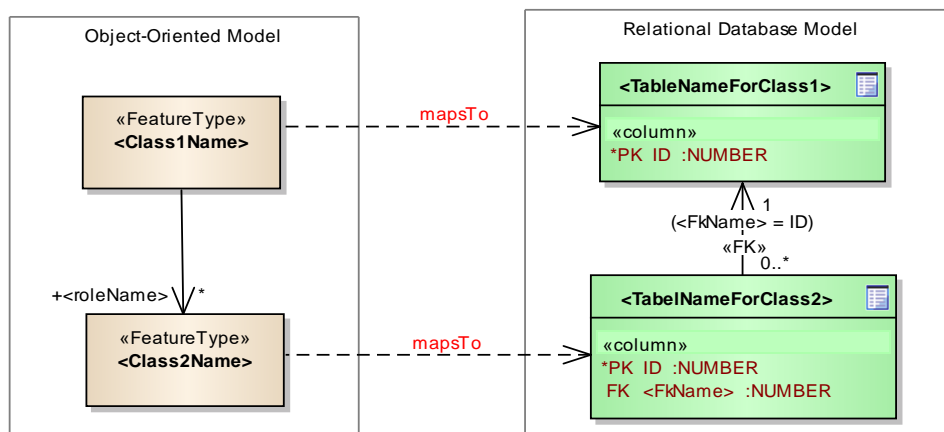


Figure 26: Mapping 1:N relationship between two different classes (variant 1)

In analogy to the mapping of the N:0..1 association, another two cases for the 1:N association can be handled through the respective mapping rules shown in Figure 27 and Figure 28.

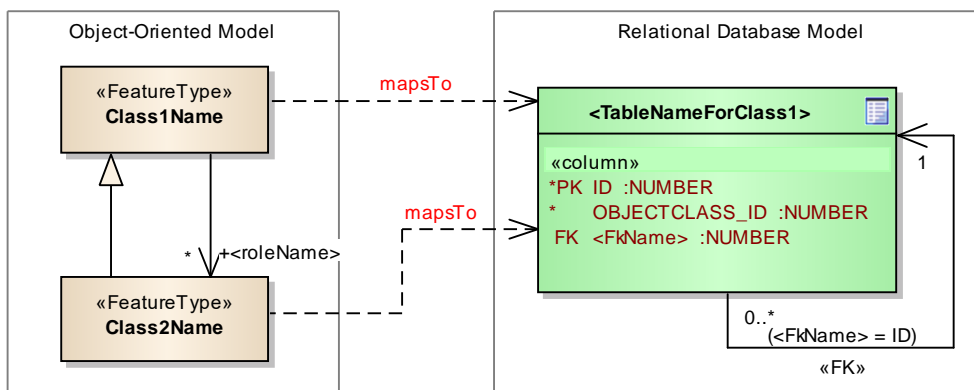


Figure 27: Mapping 1:N relationship between two classes with a shared table (variant 2)

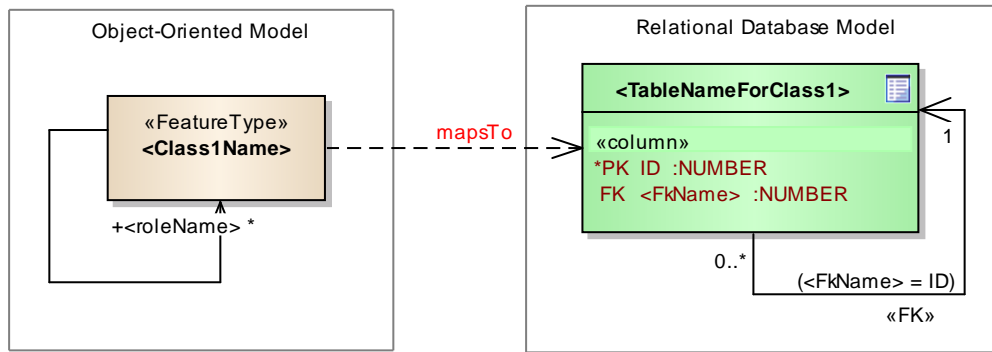


Figure 28: Mapping 1:N relationship between the same class (variant 3)

**Mapping M:N Association**

The mapping of M:N association onto relational structure requires an additional table called ‘associative table’ to link the two tables mapped from the associated classes. This associative table shall contain at least two foreign key columns which reference to the two class tables respectively. In addition, both foreign key columns can be combined to build the primary key of the associative table (cf. Figure 29).

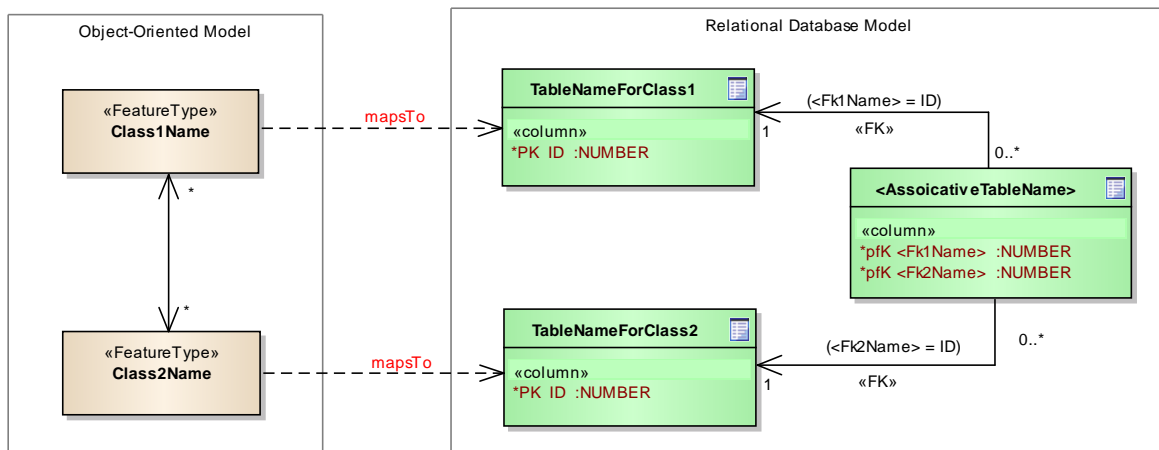


Figure 29: Mapping M:N association relationship (Variant 1)

One additional variant is that an M:N association is linked with the same class. In this case, the two foreign key columns in the associative table shall point to the primary key column of the same table (cf. Figure 30).

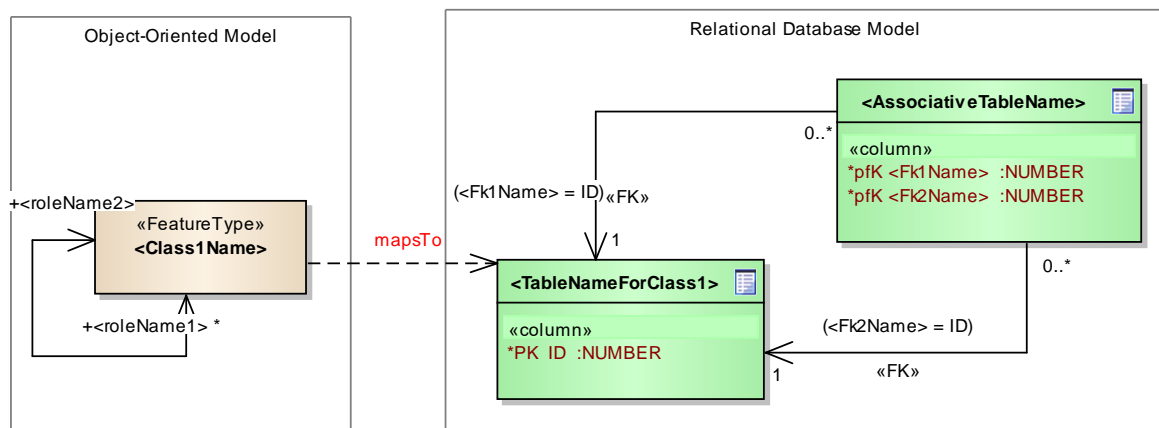


Figure 30: Mapping M:N association relationship (variant 2)

Note that the associative table can also be used for the mapping of N:1 or 1:N associations but would result in relatively lower query performance due to the additional database join operations. However, the introduction of an associative table could be very useful for dynamically extending the existing database schema. For example, when a new class is introduced which is associated with the existing class, the database schema can be easily extended by adding one table along with an associative table without needing to alter the existing class tables.

### 3.2.2 Advanced Approach for optimizing Relational Database Models

Using the standard mapping approaches outlined in the previous section, any object-oriented data model can be properly mapped onto a relational database model. However, such resulted relational database schema is usually not sufficient for some complex-structured data model like CityGML, since a large number of database joins are required when querying the data content that are distributed over many tables. To overcome this issue, a compact relational database schema with a lower number of tables is needed which can be realized based on some fine-grained optimizations by mapping multiple classes or data types onto one table. According to the 3DCityDB implementation, the following optimization approaches are summarized as follows:

#### Mapping an inheritance hierarchy onto one table

With this approach, multiple classes of an inheritance hierarchy can be mapped onto one table. For example, a table named CITYOBJECT can be used for the mapping of the GML’s class *\_GML*, and *\_Feature* as well as the CityGML’s class *\_CityObject* (cf. Figure 31). For each CityGML top-level class like *AbstractBuilding*, *AbstractBridge* and *AbstractTunnel* etc., a separate table associated with the CITYOBJECT table shall be created for storing the feature specific attributes. This way, the CITYOBJECT table can be used as a central registry of all the CityGML top-level features and allows for rapidly retrieving a list of CityObjects through a query on their common attributes like GMLID, NAME, or spatial extent ENVELOPE.

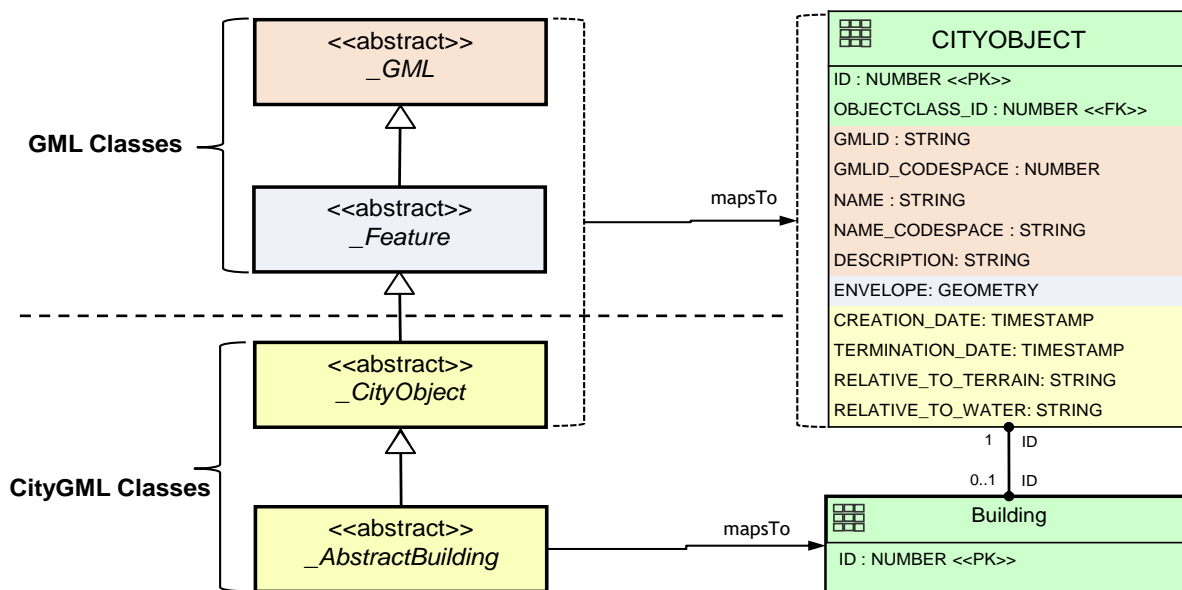


Figure 31: Example of mapping an inheritance hierarchy onto one table

### *Mapping classes at the same inheritance hierarchy level onto one table*

This mapping approach utilizes only one table to represent multiple classes which are subtyped from a common class and at the same time belong to the same inheritance hierarchy level (cf. Figure 32). This way, all the subclasses are logically mapped onto the super class table, so that the retrieval of data contents of all subclasses just needs to perform only one query on the table which allows to avoid multiple table joins to speed up the overall database performance. To distinguish the different types of instance objects stored in the table, an additional column *OBJECTCLASS\_ID* is required which can store a numeric value in each row for representing the respective class type. This type value is static and can be well documented in such a way that allows application programs to interpret the meta-information of each class automatically. This can be realized by introducing an additional table called ‘OBJECTCLASS’ whose primary key values are used for enumerating the object class IDs and referenced by the *OBJECTCLASS\_ID* columns of the other class tables. Further columns can also be added into the *OBJECTCLASS* table for storing meta-information about each class type.

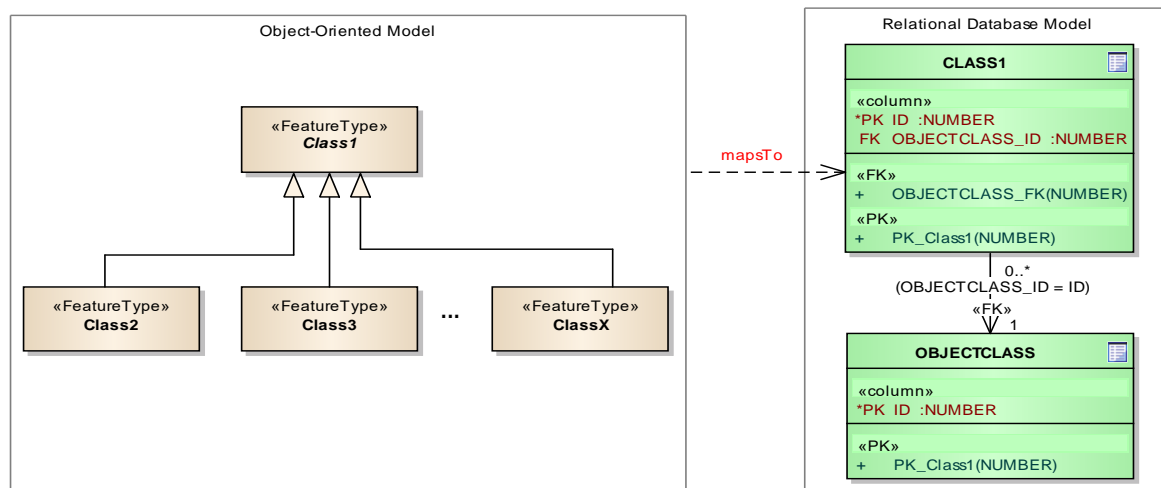


Figure 32: Example of mapping multiple classes onto one table

Analogically to the aforementioned mapping approach (cf. Figure 22), this mapping approach also has its own usage limitations. For example, if the subclasses have very different attributes or associations to other classes, a large number of empty cells will occupy the table space which can easily result in lower storage efficiency, especially when the number of subclasses is increased. Considering this situation, the utilization of this mapping approach must satisfy some conditions regarding the model definitions and structures which are given as follows.

- The super class shall be an abstract class that holds all attributes and associations which will be inherited by the subclasses.
- The subclasses shall have similar structures. Ideally, every of the subclasses shall not have any further attributes or associations with other classes.

With these conditions, the storage efficiency can be furthest retained as only one additional column storing the class IDs needs to be added. An observation of the CityGML data model shows that, this mapping approach allows to substantially improve the database performance and efficiency. For example, the thematic surfaces like wall surfaces, roof surfaces, and

ground surfaces etc. of each feature type like *Building*, *Tunnel*, and *Bridge* are abstracted to an abstract class called *\_BoundarySurface* which holds the relevant attributes and association information. For each type of thematic surface, a respective concrete class i.e. *WallSurface*, *RoofSurface*, and *GroundSurface* etc. being a subtype of the class *BoundarySurface* have been defined. This class structure exactly satisfies the outlined mapping conditions and can hence be mapped onto a compact table which allows for fast data export. For example, a typical query being usually applied is the export of a semantically rich building (LOD  $\geq 2$ ) into a 3D graphics format. In this case, the geometry information of the roof surfaces and wall surfaces can be queried using a minimum number of joins to build the 3D shape of the respective building object.

### Mapping aggregation hierarchies onto one table

In objected-oriented data models, the recursive relations of aggregated features can be properly modelled by means of a well-known design pattern called ‘*Composite Pattern*’ (cf. Gamma et al. 1995) which uses three interrelated classes (cf. Figure 33) for constructing a tree-like data structure. According to the concept of this design pattern, each instance of the class *CompositeObject* can contain an arbitrary number of, but at least one instance of the class *BasicObject* or *CompositeObject*. The *BasicObject* corresponds to the leaf in the aggregation hierarchy and shall not have child components. The conventional solution for the mapping of such data model onto relational structure is to use a foreign key for joining each object with its parent object to querying all the aggregated objects. In this case, recursive database queries must be performed which may cause high performance cost, especially if the recursion depth is unknown.

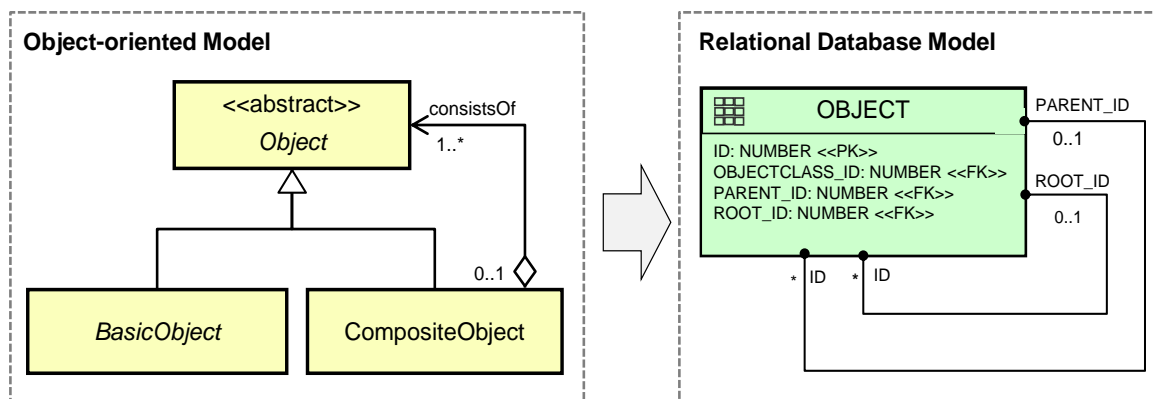


Figure 33: General idea for the mapping of an object-oriented model with the composite pattern onto an efficient relational database model

In order to guarantee good performance, a specific optimization approach has been developed which was originally introduced by (Stadler et al. 2009) and has been successfully implemented in practical applications for performing fast and efficient queries on aggregated geospatial data stored in relational databases. The key idea of this database design is, on the one hand, to utilize one database table for the mapping of all the involved feature classes along with their inheritance relationships. On the other hand, a foreign key column *PARENT\_ID* is used for representing the composition relationship. Additionally, this database table receives a foreign key column *ROOT\_ID* which holds the ID of the root element of each aggregation hierarchy. This way, the aggregation hierarchy has then to be

reconstructed in the application program, which is typically much faster than to use recursive queries on the database. Moreover, since three classes are mapped onto one table, an additional column OBJECTCLASS\_ID is required for supporting the automatic determination of class affiliation information.

**Mapping complex data types onto one table**

The optimization approach for the mapping of composite pattern can also be applied for the handling of complex data types like the B-Rep geometries such as aggregated/composite surfaces and solids (cf. Figure 34).

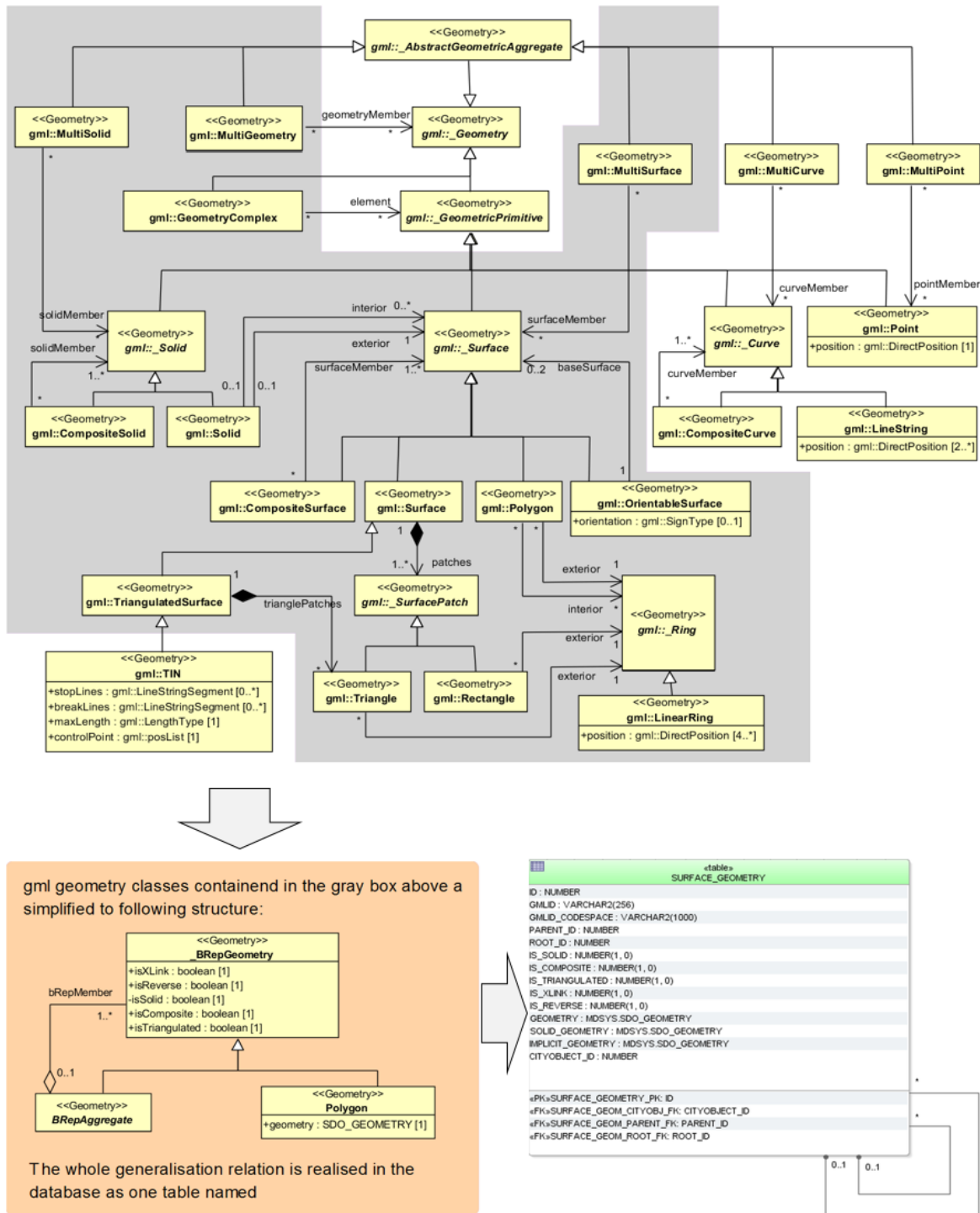


Figure 34: GML geometry types being used in the CityGML standard (cf. Kolbe et al. 2016)

With this optimization step, all surface-based geometry types can be represented in a simplified data model according to the composite pattern and consequently mapped onto a single compact table allowing for high-performance database query of all the geometry elements of an aggregation hierarchy (cf. Emgård & Zlatanova 2008). Instead of using a class ID column, the class affiliation is realized using various flag columns for characterizing the type of geometry and aggregation. For example, the IS\_SOLID distinguishes between surface and solid geometry, and the IS\_COMPOSITE can be used to determine whether this is an aggregate (e.g. MultiSolid, MultiSurface) or a composite (e.g., CompositeSolid, CompositeSurface) geometry. This approach offers a higher degree of semantic clarity of the table structure. In addition, the use of this geometry table is very simple. For example, if a feature is attributed with a MultiSurface property, a foreign key column can be added to the class table and references to the primary key column of the geometry table where the geometry data are physically stored.

### 3.3 3D City Database (3DCityDB)

#### 3.3.1 Overview

The 3D City Database (3DCityDB) is a software package allowing to import, manage, analyse, visualize, and export virtual 3D city models according to the CityGML standard. It is freely available and completely open source under the LGPL3 license earlier and switched to the Apache 2.0 licence later which allows for simplifying the inclusion and adoption of 3DCityDB in a range of third-party commercial and Open Source products. Over the past 10 years, 3DCityDB has been playing a very important role in many research and commercial projects and has been successfully employed in a number of productive environments in many cities worldwide like Munich, Berlin, Zurich, Rotterdam, Helsinki, Singapore, and London etc. In addition, numerous companies around the world deploy the 3DCityDB at the core of their commercial products and services to create, maintain, visualize, transform, and export virtual 3D city models for building a 3D geospatial infrastructure and functionality rich GIS applications. For example, the state mapping agencies of all 16 states in Germany store and manage the state-wide collected 3D building models in CityGML LOD1 and LOD2 using 3DCityDB.

The development of 3DCityDB was started back in 2003 by the research team of Thomas H. Kolbe and Prof. Lutz Plümer at the Institute for Cartography and Geoinformation at University of Bonn. Since today, the 3DCityDB has constantly evolved and more and more functionalities have been added in the context of many research activities. During that period, the 3DCityDB has been widespread worldwide and become a key platform in demonstrating the practical usability and versatility of CityGML. In 2012, the developer team (by that time based at TU Berlin) received the *Oracle Spatial Excellence Award for Education and Research* from *Oracle USA* for the work on 3DCityDB. Since 2013 the 3DCityDB and its tools have been further developed by the Chair of Geoinformatics of TU Munich (TUMGI) lead by Prof. Thomas H. Kolbe in collaboration with the companies virtualcitySYSTEMS GmbH and M.O.S.S. Computer Grafik Systeme GmbH on the basis of a cooperation agreement to facilitate the continuation of the improvement and development of 3DCityDB. In the same year, the author of the thesis started with his PhD at TUMGI and joined the



3DCityDB team as one of the major developers to work on the conceptual design and technical implementation of the 3DCityDB software tools.

### 3.3.2 3DCityDB Insights

Basically, the 3DCityDB software package consists of a relational database schema along with a set of database procedures and software tools (cf. Figure 35), which allow to efficiently import, manage, analyse, visualize, and export virtual 3D city models according to the CityGML standard. The database schema is the core component of the 3DCityDB for storing CityGML datasets in a central repository. According to the mapping rules introduced in the previous subsection, the database schema results from a mapping of the object-oriented data model of CityGML onto a relational structure on top of the spatially-enhanced relational database management systems (SRDBMS) and can be operated using either the commercial Oracle or the Open Source PostgreSQL RDBMS with their spatial extensions. In this way, CityGML data can be easily accessed by external GIS and ETL software applications or the provided database procedures to e.g. enrich a 3D city model by adding information to the corresponding database tables programmatically. Moreover, the data contents stored in the 3DCityDB database can be flexibly extracted and exchanged by exporting a 3D city model to a CityGML document from the database without any information loss.

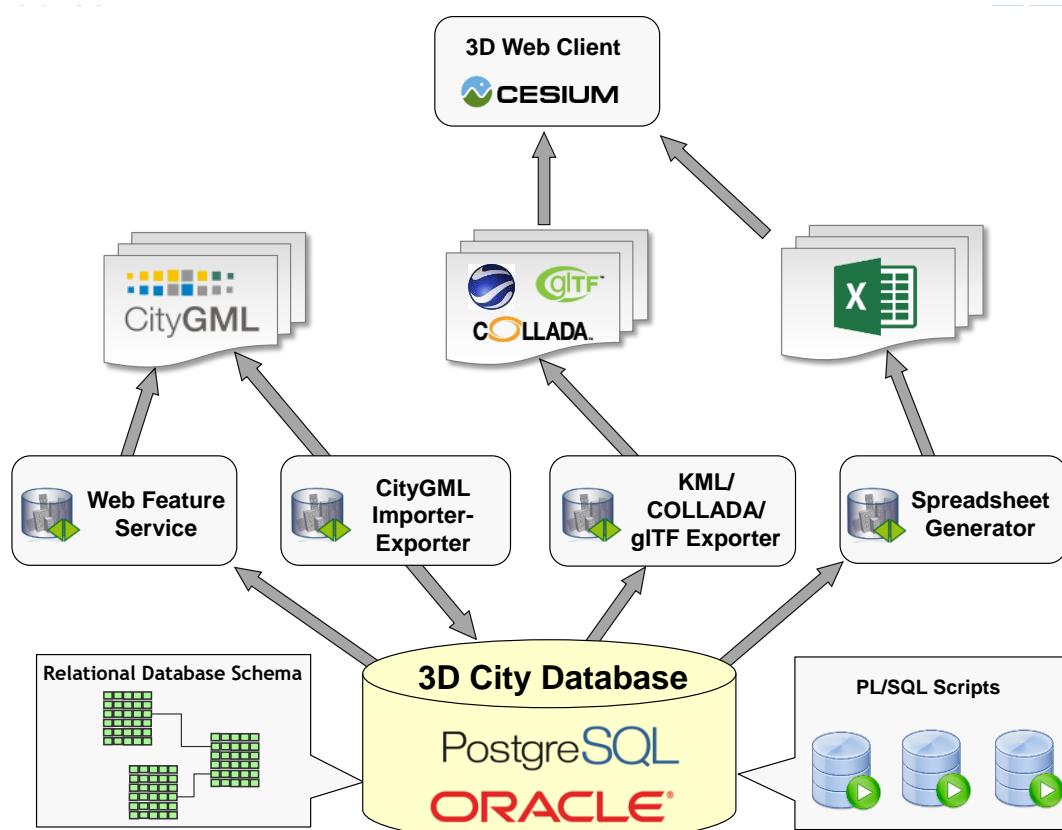


Figure 35: Relevant components of the 3DCityDB Software Suite

The first important 3DCityDB software toolkit are the database procedures which can be automatically installed at the database side during the setup procedure of a 3DCityDB database instance. They are written in the structured query language PL/SQL for Oracle and in PL/pgSQL for PostgreSQL. Since some of these procedures expose some similar



functionalities, they can hence be managed into six packages, namely SRS, STAT, INDX, ENVELOPE, and DELETE (cf. Figure 36). The package SRS mainly provides a useful function to transform CityGML's spatial data into another coordinate system for the output document. The package STAT can be applied to count all entries in all data tables and generate a report listing the number of rows in the individual data tables. The package DELETE consists of several functions allowing to delete single or multiple city objects from the database according to the given row ID or an object class ID in the CITYOBJECT table. Each function automatically takes care of the integrity constraints between the related tables to properly clean up the corresponding data contents. The package ENVELOPE provides functions for calculating the minimum 3D bounding volume of a city object according to its geometry contents and also allows for updating the ENVELOPE attribute of the respective city object with the calculated value. In order to ensure data consistency, it is hence very important to run this function whenever one of the geometry representations of a city object has been changed. The package INDEX contains the function for activating and deactivating the ordinary indexes and spatial indexes on those columns that are frequently used for performing queries. This allows to deactivate the spatial indexes before running a CityGML import on a big amount of data and to reactivate the spatial indexes afterwards. This way, the import process will run much faster than with enabled spatial indexes. The last package UTIL offers various utility functions i.e. checking database version information, performing affine transformation on the 3D coordinates, determination of the mapping relationships between 3DCityDB tables and CityGML classes.

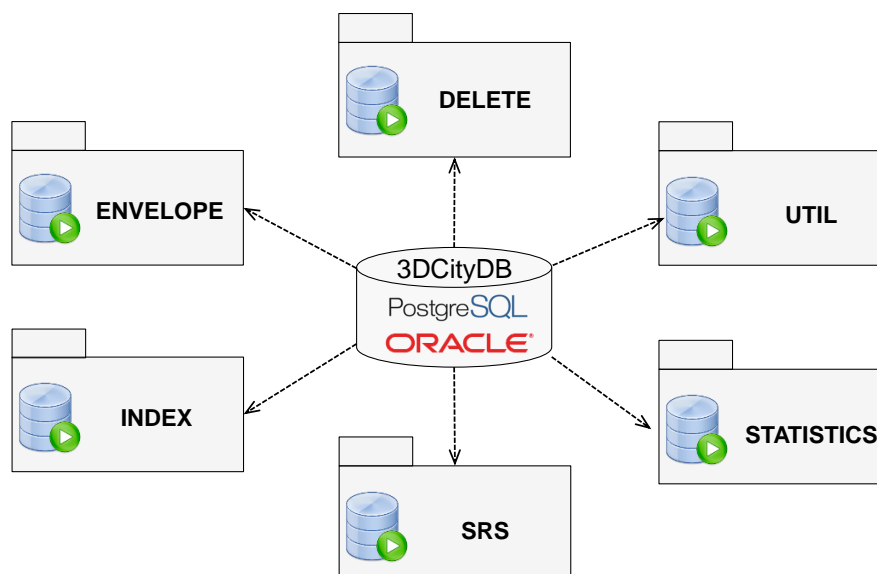


Figure 36: Overview of the 3DCityDB database procedure packages

One of the major software tools included in the 3DCityDB software toolkit is the Importer/Exporter which is a Java-based desktop application serving as a front-end for the 3DCityDB database for high-performance reading and writing of large CityGML datasets with arbitrary file size. For reading and writing CityGML documents, a low-level Java API called `citygml4j` is available which provides a convenient way for processing and validating CityGML datasets against the CityGML, GML as well as the xAL schema definition files. This is realized by using the Java's XML Schema binding compiler (`xjc`) to compile the CityGML, GML, and OASIS xAL models to a set of corresponding Java classes which are

kept static and provide an object-oriented view for handling CityGML features along with their properties in Java during runtime. While performing the CityGML import process (cf. Figure 37), each CityGML top-level feature element is read chunk-wise and automatically transformed to a Java object according to the corresponding class definition in the citygml4j context. All these Java objects are organized as a buffered queue and can be successively imported into the database concurrently by means of a multi-threaded approach to increase the overall processing performance. In order to make full use of hardware CPUs/cores and to avoid the thread lifecycle overhead, a thread pool is employed to manage and control the number of the threads according to the number of the available processors of the hardware being used.

The last step of the import process is resolving of the XLinks information in the CityGML datasets (cf. Figure 37). This addresses an issue that a CityGML feature or geometry could be referenced by other ones using the GML XLink mechanism. Since some CityGML objects in the beginning of a CityGML file can point to objects located at the end of the same CityGML file, resolving the XLinks usually requires reading the entire CityGML data into the main memory, which can easily cause memory overflow issues when dealing with very large CityGML datasets ( $\gg 4\text{GB}$ ). To overcome this issue, CityGML features and geometries are first read and imported, neglecting all the XLink reference information which shall, however, be temporarily stored into the database. When the first import process is done, the XLink reference information stored in the database will be resolved again and written into the corresponding CityGML data tables to complete the entire CityGML import process.

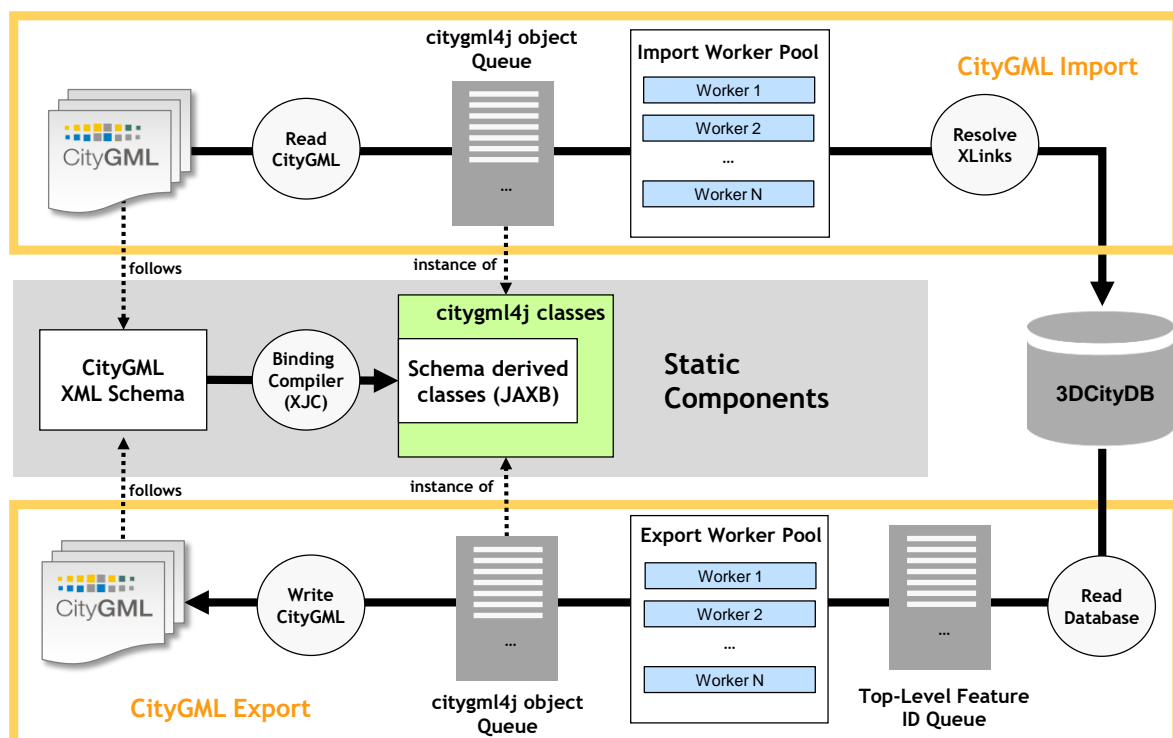


Figure 37: Software structure of the CityGML Import/Export Tool (cf. Stadler et al. 2009)

During the export process (cf. Figure 37), a list of GMLIDs of the top-level features satisfying the user-defined filter criteria i.e. feature class types and geographic bounding box etc. are first queried from the database. In the subsequent step, a worker pool containing a number of worker threads is constructed and each GMLID is processed by a worker thread for creating a

citygml4j object from the CityGML feature content queried from the respective database tables. In the last step, the citygml4j objects are marshalled to XML elements and written to a CityGML instance document.

Although the CityGML Importer/Exporter offers extensive functionalities for reading and writing CityGML documents from the database, it is however only applicable on a desktop environment and does not allow for accessing the data using a platform-independent call. To overcome this limitation, the 3DCityDB offers a web service implementation by extending the core modules of the CityGML Import/Export tool to allow for web-based access to the 3D city objects stored in the database. It is realized based on the OGC Web Feature Service (WFS) Interface Standard which provides a standardized and open interface for requesting geographic features across the Web using a platform-independent HTTP request. Thus, the 3DCityDB users are no longer limited to using the Importer/Exporter tool for the data retrieval but can also directly use the WFS interface via a web browser or a WFS-aware application.

The 3DCityDB WFS is implemented as a *Java web application* based on the *Java Servlet* technology and must hence be run in a Java servlet container e.g. Apache Tomcat on a web server. The workflow of executing a WFS procedure is illustrated in Figure 38. When a WFS client sends a request to the WFS server to retrieve certain CityGML features, the 3DCityDB WFS servlet will first capture and parse the request information and translate it to a corresponding database query to obtain a list of GMLIDs of the CityGML top-level features that satisfy the filter criteria encoded in the WFS request messages. These feature IDs will be then handed over to the CityGML Import/Export module which utilizes its pre-compiled citygml4j/JAXB classes as well as the multi-threading API for efficiently querying and generating the corresponding CityGML XML elements. Finally, these XML datasets will be returned as a response of the WFS request and can be directly downloaded or visualized in a WFS client.

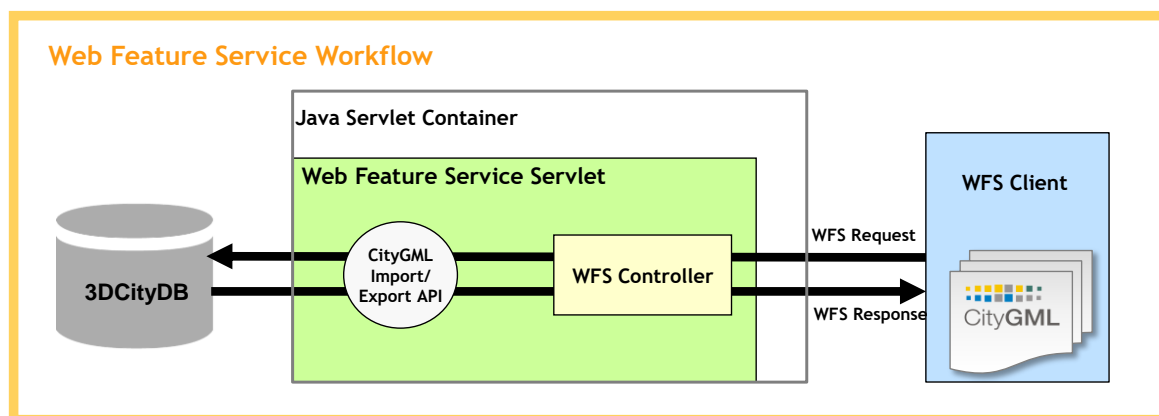


Figure 38: Implementation of the 3DCityDB Web Feature Service

The CityGML Import/Export tool also comes with an extensive plugin API which allows for dynamically extending the existing Import/Export functionalities to support the modular development and deployment of additional tools for interacting with the 3D City Database or external datasets. Using the plugin API, software developers are able to create their own plugins which can be separately added to the Importer/Exporter. A plugin can be easily installed by copying its mandatory files including the compiled JAR file and related libraries

etc. into a specific subfolder of the Importer/Exporter installation directory and will be automatically found and launched when starting up the Importer/Exporter. In addition to extending the functionalities, a plugin may also extend the GUI of the Importer/Exporter by providing its own user dialog that will be embedded and rendered in the main operations window of the Importer/Exporter.

Per default, the 3DCityDB provides an internal plugin called KML/COLLADA/glTF Exporter which is delivered together with the Import/Exporter. Using this plugin, the spatial data contents of CityGML features can be directly exported using popular 3D visualization formats like KML, COLLADA, and glTF for high-performance visualization and exploration in a broad range of 3D mapping applications or Earth browsers like Google Earth, ArcGIS Explorer, and Cesium. This export process (cf. Figure 39) follows a similar logic to that of the CityGML exports. In the first step, according to the user-defined filter criteria, the GMLIDs of the satisfied features are first queried from the database and then passed to a worker pool which is implemented using the same multi-threading API of the Import/Export tool. Depending on the hardware being used, this worker pool is able to dynamically create an optimal number of worker threads each of which is responsible for taking one GMLID after another from the waiting queue and querying the respective feature content from the database for creating a KML/COLLADA java object. The class definition of the java object is pre-generated by means of the XML Schema binding compiler (xjc) for compiling the KML/COLLADA's XML schema definition files to the corresponding Java classes which allows for directly marshalling the created java objects to the corresponding XML elements using the JAXB library. In addition, the creation of the glTF models is done through a separate processing step which utilizes a third-party tool called *Collada2glTF* for converting the COLLADA to glTF models. Moreover, in order to achieve sophisticated streaming and rendering performance in a 3D viewer, the KML/COLLADA/glTF models can be created with various display forms with different level of details organized in an efficient spatial data structure when exporting a large 3D city model. Further details are elaborated in chapter 5.

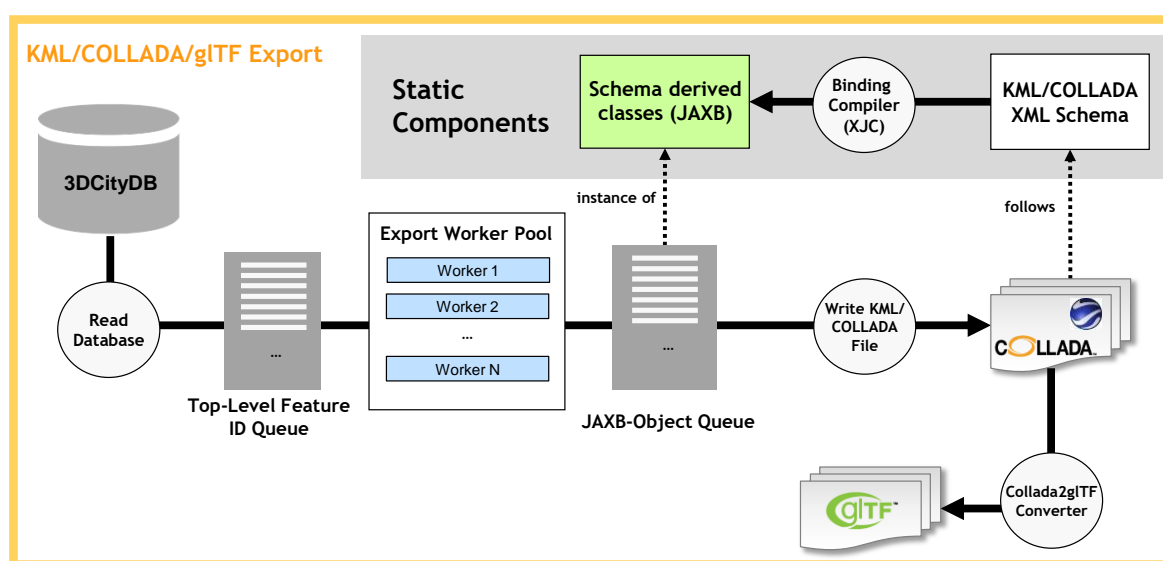


Figure 39: Workflow of generating KML/COLLADA/glTF visualization models

The 3DCityDB offers another plugin called *Spreadsheet Generator* plugin which is an external plugin shipped with the Importer/Exporter and can be installed optionally. This

plugin can export the thematic contents of a 3D city models from a 3DCityDB instance to a table format, either to a CSV or a Microsoft Excel file, where the first column lists the unique identifiers (GMLIDs) of the exported city objects each of which refers to one spreadsheet row. The generated spreadsheets can be opened using a spreadsheet application like Microsoft Excel and Open Office Calc etc. or uploaded to a Cloud-based online spreadsheet service e.g. Google Drive and Microsoft OneDrive which allows for interoperable web access by multiple users. In addition, the relevant features of spreadsheet programs like formula calculation and graphing tools, are applicable to the exported thematic data for realizing various analysis and simulation functionalities in web-based 3D GIS applications. More details are illustrated in chapter 6.

The spreadsheet generation process (cf. Figure 40) is similar to the workflow of the KML/COLLADA/glTF exporter and is also implemented based on a multi-threading programming by means of the concurrency API provided by the Importer/Exporter. In this case, each thread in the worker pool is dedicated to query the thematic contents of a top-level feature and map the results onto a table row based on a so-called table mapping template which can be freely defined by the users. This template is a text-based file and comprises a set of key-value pairs (KVP) each of which can be seen as a spreadsheet column definition: The “key” of a KVP specifies an expression which can be directly translated to a SQL statement for fetching the data from a column of a specific table in the 3DCityDB, whereas the “value” specifies the column name in the output spreadsheet. With this template file, the value of each spreadsheet’s column can be dynamically queried from the database for every city object and written to the spreadsheet at the export time. In case that more than one value is returned by a query expression, it is possible to select the first or the last one of the returned values or simply group them as a comma separated string value which will be then passed into the corresponding spreadsheet cell.

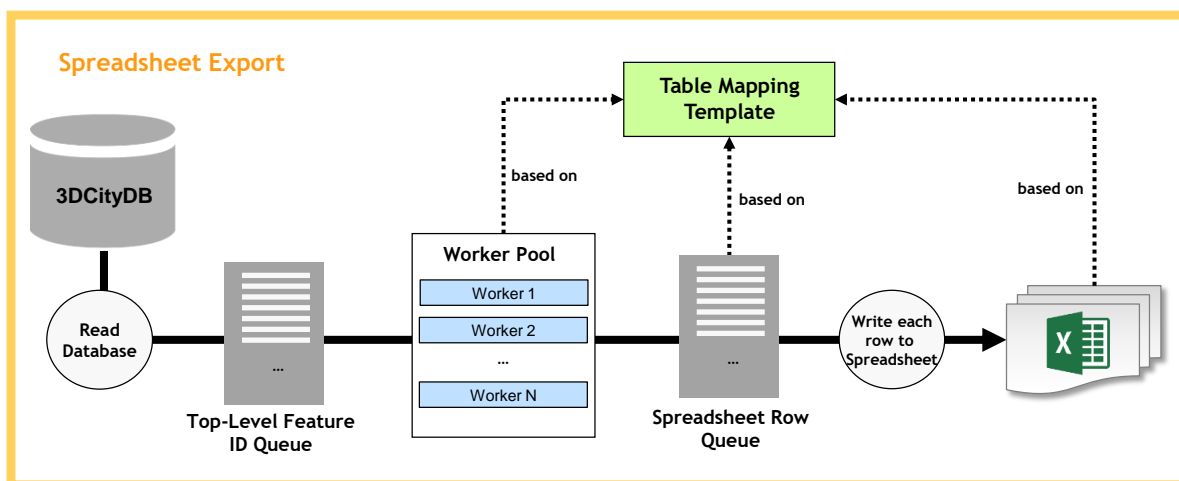


Figure 40: Workflow of generating spreadsheet from 3DCityDB

Starting from version 3.3.0, the 3DCityDB software package comes with a new tool called *3DCityDB-Web-Map-Client* or simply called *3D web client* acting as a web-based front-end for high-performance 3D visualization and interactive exploration of arbitrarily large semantic 3D city models (cf. chapter 5). Basically, the 3D web client has been developed based on the open-source Cesium virtual globe which provides high-performance and cross-platform functionalities like displaying 3D geographic contents on the web without the need to install

additional web browser plugins. While developing the 3D web client, various extensions have been made to the Cesium 3D viewer in order to facilitate users to view and explore 3D city models conveniently. The major one among those extensions is the high-performance interaction and visualization of very large 3D models which are exported using the KML/COLLADA/glTF Exporter and can be published via an in-house or a Cloud-based webserver for enabling the data access over the Internet (cf. Figure 41). Furthermore, these 3D visualization models can be logically linked with multiple Cloud-based online spreadsheets containing the thematic data exported using the Spreadsheet Generator Plugin and allows for interactively querying the attribute information of every city object in addition to the pure 3D visualization by means of the Cloud-based service like Google Docs (cf. Herrerueta et al. 2012, Yao et al. 2014).

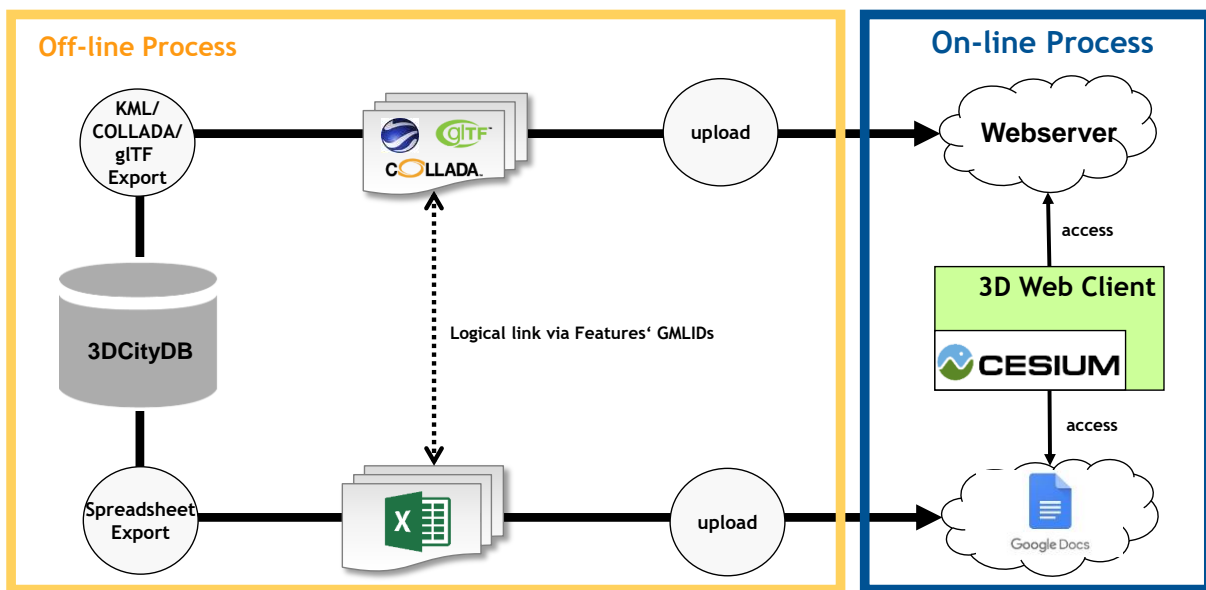


Figure 41: Workflow of using 3DCityDB web client coupled with Cloud-based online spreadsheets

In summary, with the help of the presented relational database schema, database procedures, as well as the front-end software tools, the 3DCityDB has become an extensive open-source GIS software ecosystem with the full functionalities ranging from efficient storage and management of virtual 3D city models according to the CityGML standard, via the dynamic interaction with the spatial and thematic data, up to high-performance visualization and exploration of them on the web. In combination with other GIS, BIM, and ETL software systems, various applications can be built to accomplish certain tasks for different application domains. Further details about the utilization of such software systems along with the semantic 3D city models of CityGML in practical applications are comprehensively discussed in chapter 6.

## Chapter 4 Management of Domain Extendable 3D City Models

Although CityGML offers a semantically rich 3D city model that covers the most relevant geographic phenomena in real world, various additional feature classes and heterogeneous attribute properties are usually required in some specific application cases like urban planning, energy simulations, utility network analysis, and noise propagation calculations etc. For instance, the heat demand calculation for a building typically needs the information about the construction materials of the surrounding walls in order to determine the thermal resistance about how easily the heat can flow out the building. Another example are utility network constructions like water pipes and electrical connectors etc., which shall be represented in a graph structure for facilitating network analyse. However, all such information are currently not included in the existing CityGML data model. Concerning this situation, the 3D city model must be extendable and can thereby be dynamically enriched with arbitrary domain-specific information models. This is conceptually realizable since each 3D city model object is usually assigned with a unique identifier, which must be kept stable throughout the lifetime of the referenced real-world object. Using such identifier, different domain-specific information can be properly joined and attached to the same object within a common framework like CityGML. Consequently, the extended 3D city model will become a very sophisticated data basis that can be efficiently managed and exchanged across different application systems.

To realize the efficient management of such domain extendable 3D city models, numerous research and implementation challenges have been identified concerning the model development and management in practical applications. First of all, a systematic extension mechanism shall be chosen which should allow developers or organizations to develop their domain-specific data models on their own and dynamically add the extension models to the CityGML data model. The extended data model must be organized in such a way that can ensure the interoperable data exchange across different application domains without information loss. Second, the extended 3D city models must be managed as platform-independent models such that they shall only be modeled on the conceptual level and can be automatically converted to various encodings in different formats depending on the application needs. For example, the automatic generation of a relational data model is of great importance for creating a relational database schema to realize the efficient data storage in a relational database management system. Third, from the perspective of industrial application, the software tools for the data interaction like import and export must also be extendable to facilitate the manipulation of instance datasets of the extended data models over the database systems.

For overcoming the challenges mentioned above, a number of research and implementation results have been successfully obtained during the dissertation work and the main contents are structured as follows: the first section reviews the relevant concepts for the modelling and development of domain-extendable 3D city models based on the CityGML standard using its built-in extension mechanism called 'ADE' which serves as the key foundation of the entire chapter. Regarding the data management of the extended 3D city models, a conceptual idea of developing a dynamically extendable database framework based on the existing 3DCityDB solution is given in order to make full use of the capability of the 3DCityDB for handling the standard CityGML data model. In the second section, more focus is put on the answering the



question of how to automatically derive 3DCityDB-compliant relational database schemas for arbitrary CityGML ADEs. To reach this purpose, the existing solutions have been carefully examined to identify their drawbacks and limitations in the automatic derivation of compact relational database structures. Based on this survey, a new advanced approach has been developed which is based on ‘Graph Transformation Systems’. Finally, a technical realization of the proposed approach has also been carried out and the corresponding details of the implementation along with a representative example are presented in the last section to prove the feasibility and advantages of the developed approach compared to previous software solutions.

#### 4.1 Extending the CityGML Data Model

As mentioned in section 3.1.2, there are two major ways for extending the standard CityGML data model by augmenting the existing CityGML feature classes with additional domain-specific attribute information or introducing new types of city objects: one is to use the *Generics* module which is a standard thematic module defined in the CityGML specification. The other one is the utilization of CityGML’s extension mechanism called ‘Application Domain Extension’ (ADE), which requires that extra application schemas must be defined and linked to the CityGML’s standard application schemas. Compared to the latter one, the former one provides a relatively easy way as there is no need to change the existing CityGML application schema. For example, it is very easy to use the *GenericCityObject* feature class to represent any alien city objects with respect to their relevant thematic and spatial properties. Moreover, in case of adding domain-specific attribute information to a city object, *GenericAttributes* can be used for attaching an arbitrary number of attributes with different data types to the existing CityGML feature classes. However, according to the CityGML specification, the approach of using the *Generic* module also has its own limitations and disadvantages:

- The class *GenericCityObject* is defined with a very simple structure and is hence not able to describe the semantic relationships like association, aggregation/composition, or specialization/generalization between feature classes, which might be associated with or inherited from the existing CityGML classes. Besides, the generic attributes only support a limited set of primitive data types like string, integer, floating-point number, and date etc. or an aggregation of them, but is not able to represent those data types that have complex structures like inheritance relationships. Obviously, these two limitations hinder the use of the *Generics* module from representing the complex-structured feature classes which can frequently occur in practical application cases.
- The utilization of generic attributes lacks the possibility to formally specify the minimum or maximum occurrence of generic attributes, since they are allowed to occur an arbitrary number of times in a city object according to the CityGML standard. Consequently, it is impossible to use an XML-aware program to guarantee the attribute multiplicities by validating the instance documents against the CityGML schema definition files in a formal way. Thus, in order to ensure the semantic interoperability of the extended data models, the attribute validation procedure can only be done by the CityGML reader applications which will however require extra implementation efforts.



- Another disadvantage is that the generic attributes or city object belonging to different application domains are not allowed to have the same name in an instance document simultaneously, because no additional attribute like ‘Namespace’ aligning with the attribute name is available which allow users or application programs to distinguish the domain affiliation. As a result, the compatible use of different domain-specific information cannot be guaranteed and naming conflict issues can easily happen.

Concerning these usage limitations of the CityGML Generics module, the solution of using the CityGML ADE mechanism has been chosen as the foundations of the thesis for exploring how to realize the efficient management of domain-extendable 3D city models within the CityGML framework. As the entry point, the following subsection will first give an insight into the key concept of the CityGML ADE.

### 4.1.1 CityGML ADE Insights

According to the CityGML standard, a CityGML ADE is actually a GML application schema for modelling domain-specific information which can be incorporated into CityGML instance documents. Hence, it has very close relations with the CityGML and GML specification as well as the ISO 19100 standard family. These relations are illustrated in Figure 42.

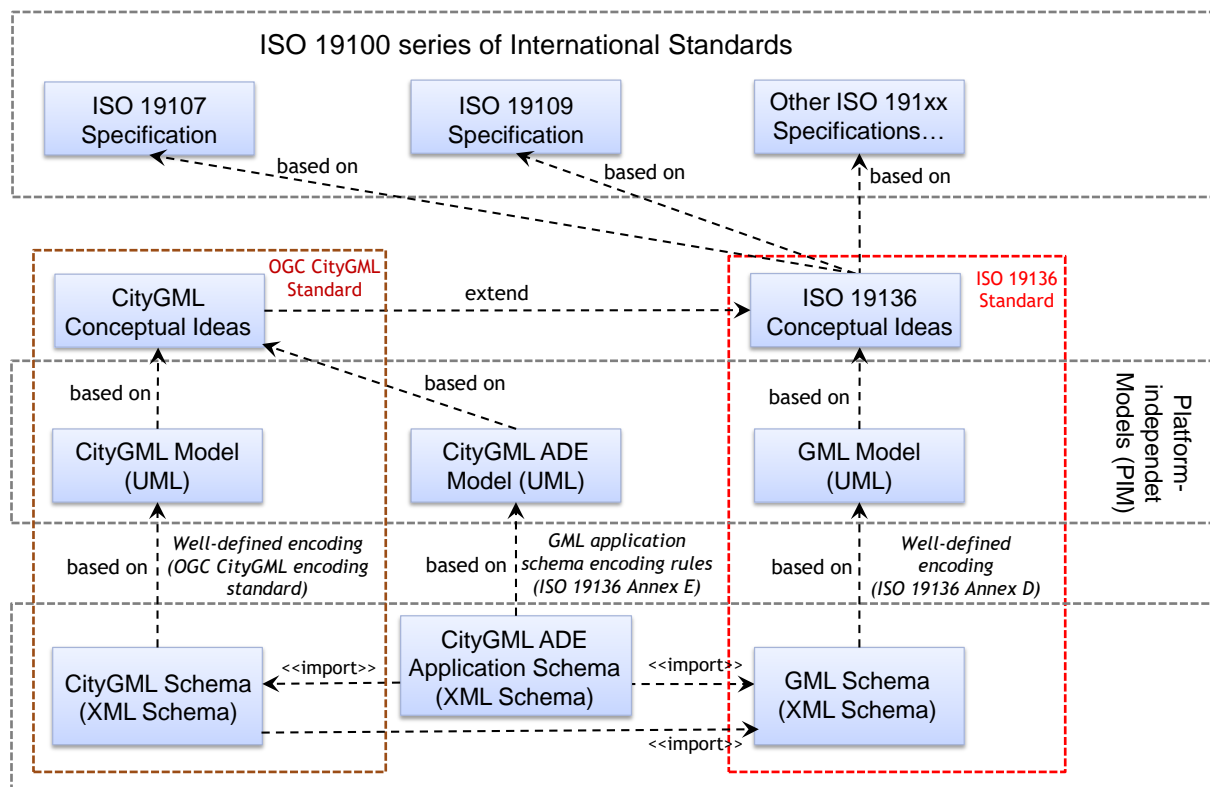


Figure 42: Relationships between the ISO 19100 standard family and CityGML ADEs

According to chapter 2.4, the ISO 19136 standard “Geography Markup Language (GML)” provides an open and manufacturer independent framework for the definition of geospatial data models and is completely based on the ISO 19100 standards such as the ISO standard 19109 “Rules for Application Schema”, the ISO standard 19107 “Spatial Schema”. In addition, the ISO 19136 standard provides an extensive UML profile together with a

normative UML-to-XML encoding rule set which forms an MDA-compliant framework allowing for the development of a variety of application schemas based on a standardized exchange interface using GML standard. As an application schema of GML, the CityGML data model specifies the application specific concepts for the spatio-semantic modelling of 3D city and landscape objects. It can be maintained as a platform-independent information model using UML models and has also been implemented as an XML-based schema in compliance with GML schema. In analogy to the standard CityGML model, CityGML ADEs can also be originated from the start with the design of a UML diagram (cf. van den Brink et al. 2013). Using the UML-to-XML encoding rules defined in the ISO 19136 standard, a CityGML ADE can also be implemented as an XML schema definition file that references both the CityGML and GML schemas. Note that although the UML model can be directly serialized to a text-based document using the standard format “XML Metadata Interchange (XMI)” for the purpose of information exchange between different application systems, CityGML ADEs are nowadays mostly transmitted using XML schema files. This is because, the XML schema provides a sophisticated syntactical structure which is not only able to represent the semantics of object-oriented models with complex data structures and can also be easily interpreted and parsed by many XML-schema-aware software tools or programming libraries. Moreover, the XML instance documents can also be checked against their meta-models directly using the respective XML schemas to ensure the validity of the instance documents of CityGML ADEs.

#### 4.1.2 Development of CityGML ADEs

According to the CityGML standard, there are two approaches for developing a CityGML ADE through extending the existing CityGML data models:

- **Defining new ADE classes:** Depending on the application needs, new classes usually have to be introduced which are not predefined in the CityGML specification. Since CityGML is based on the GML standard, each ADE class shall be derived from GML’s top-most abstract class ‘*\_GML*’ or its subtype ‘*\_Feature*’ which both are used for representing identifiable real-world objects. Alternatively, since CityGML is based on the object-oriented paradigm, new ADE classes can also be defined by subclassing the existing CityGML classes and augmenting them with additional domain-specific properties to construct an extended version of the corresponding CityGML superclass. For example, a new class called ‘*IndustrialBuilding*’ can be defined as a subclass of the CityGML class ‘*\_Building*’ for explicitly representing the industrial constructions like the factory workshops or machine halls. Although both approaches of defining new ADE classes are feasible, it is however highly recommended to subtype the CityGML class as far as possible when defining a new ADE class, since this new class can transitively inherit all the properties and relations from its superordinate classes e.g. *\_CityObject* whose instances can be treated as city object members of a *CityModel* object directly which is a root element in a CityGML instance document. Furthermore, multiple industrial building objects can also be grouped as a *CityObjectGroup* object by making use of the CityGML’s grouping concept (cf. chapter 3.1.2).
- **Adding ADE properties to existing CityGML classes:** In CityGML, a feature class can be dynamically enriched with arbitrary number of additional properties of any data types. This is realized using the XML substitution mechanism according to which

each CityGML feature class is equipped with an interface attribute called “hook” which has an unlimited multiplicity and can be substituted by any elements defined in the XML schema definition file. In this way, model developers are able to extend an existing feature class by defining new properties and inject them into the target feature class. Compared to the first approach presented above, this approach has the advantage that there is no need to define a new subclass and the extended CityGML class can still be handled by the existing CityGML applications which just need to simply ignore the ADE attributes while parsing the CityGML instance document. However, this approach violates the concept of the object-oriented modelling in UML as it does not support the concept of injecting properties to a well-defined class. To overcome this issue, (van den Brink et al. 2013) introduce a workaround solution which is based on the UML subtyping expression and utilizes two specific stereotypes `<<ADEElement>>` and `<<ADE>>` to mark the subclass and specialization relation respectively. In this way, all properties belonging to the `<<ADEElement>>` class will be recognized as the ADE properties which shall be injected into the corresponding parent CityGML class.

In the past years, numerous CityGML ADEs have been developed using the software tool *Enterprise Architect* (EA) which is a UML design tool fully supporting the MDA-based development of GML-compliant data models. The UML stereotypes used in the ISO 19100 standards as well as the GML and CityGML data models have also been implemented based on the EA-framework to facilitate the convenient design of CityGML ADEs at the conceptual level. In order to derive the GML application schema of an ADE, the open-source software tool *ShapeChange* (cf. ShapeChange 2017) can be used to read and parse the EA-model of a given CityGML ADE and automatically create the corresponding XML schema definition file according to the UML-to-XML encoding rules defined in the ISO 19136 standard. Concerning the creation of instance datasets of a CityGML ADE, the software ‘Feature Manipulation Engine (FME)’ of Safe Software is a suitable tool which provides a sophisticated transformer utility called “XMLTempater” that allows to create valid XML datasets of an ADE with respect to its XML schema definition file. The universal reader functions provided by the FME allow to read the geospatial data in different formats and pass the interpreted data into the corresponding ADE datasets according to an explicit definition of the semantic mapping between the input and output data.

### 4.1.3 Extending the 3DCityDB for CityGML ADEs

To store and manage CityGML datasets with ADE data for interoperable data access, the spatially enhanced database management system is considered to be the most important solution. As mentioned in the previous chapter, the open source relational database solution 3D City Database (3DCityDB) has been designed for the high-performance storage, management, and analysis CityGML datasets despite their complex data structures and spatial characteristics. However, the 3DCityDB database schema didn’t support the storage of CityGML ADEs since it was resulted from a manual derivation of the CityGML data model to a fixed relational database structure which cannot handle the data elements being not from the standard CityGML data model. Therefore, the 3DCityDB must be extended to become

elastic in order to be able to support any CityGML ADEs which may also have very complex model structures.

To realize this, there exists a traditional generic solution which employs a set of tables with a rigid relational structure allowing to map arbitrary XML-based graph like structures onto a relational database model (cf. Florescu & Kossmann 1999, Li et al. 2004). This approach has been proven to be a relatively applicable solution for handling arbitrary XML/GML documents. However, this approach also poses two significant drawbacks. First, the mapped relational structure requires a large number of recursive joins for the representation of aggregation and inheritance hierarchies of the object-oriented data models which will result in lower database performance. Second, it lacks the explicit mapping between classes and tables, since all classes and their attributes are mapped onto a mixed table structure which makes applications difficult to retrieve the data contents from the database. Thus, instead of a generic database structure, the 3DCityDB database models must be elastically extendable in such a way that each CityGML ADE shall be handled like a “plugin” of the 3DCityDB to allow for dynamically extending the kernel 3DCityDB database schema (cf. Figure 43). However, since CityGML ADEs can be defined with very complex model structures in practical situations, a challenging task is to find a way to automatically derive 3DCityDB-like database schemas for supporting the high-efficient storage and maintenance of arbitrary CityGML ADEs. However, due to the fact that the underlying design decisions of the 3DCityDB strongly rely on many manual steps, e.g. recognizing a certain complex model structure and mapping it to a particular target database structure, which makes the automation of the ADE mapping process much harder than the generic solution and hence requires further investigations and research work.

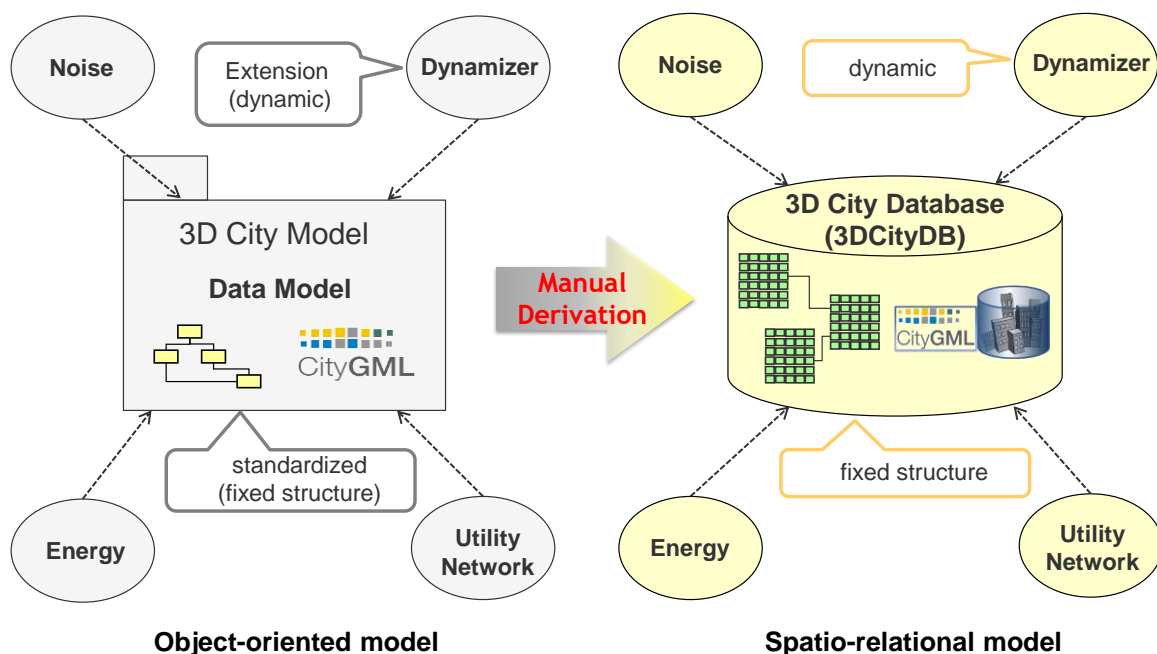


Figure 43: Conceptual workflow for creating a dynamically extendable 3D geo-database for CityGML

## 4.2 Automatic Derivation of Relational Database Schemas for ADEs

As mentioned in the previous section, CityGML ADEs are usually defined using XML schema definition files which shall be used as the input asset for performing the automatic

derivation of the 3DCityDB-compliant relational database schemas (cf. Figure 44). While the XML schema of a CityGML ADE natively represents an object-oriented data structure, the target database schema has a relational table structure which must be the result from a model transformation process to execute the conversion of an object-oriented model as the input to a relational database model as the output (cf. Mens & Van Gorp 2006, Bohannon et al. 2002) using a computer-aided transformation system. To realize this, both the input and output models have to be mapped onto certain kinds of computer-interpretable formats such that the model transformation process can be automatically carried out by applying a set of user-definable mapping rules (cf. Ng & Learmont 2002). In the subsequent step, the output model containing the relational database structure can be parsed and translated into the corresponding database schema which shall be represented as specific SQL scripts that can be directly used for the creation of database schemas for different types of database products e.g. Oracle Spatial and PostgreSQL/PostGIS etc.

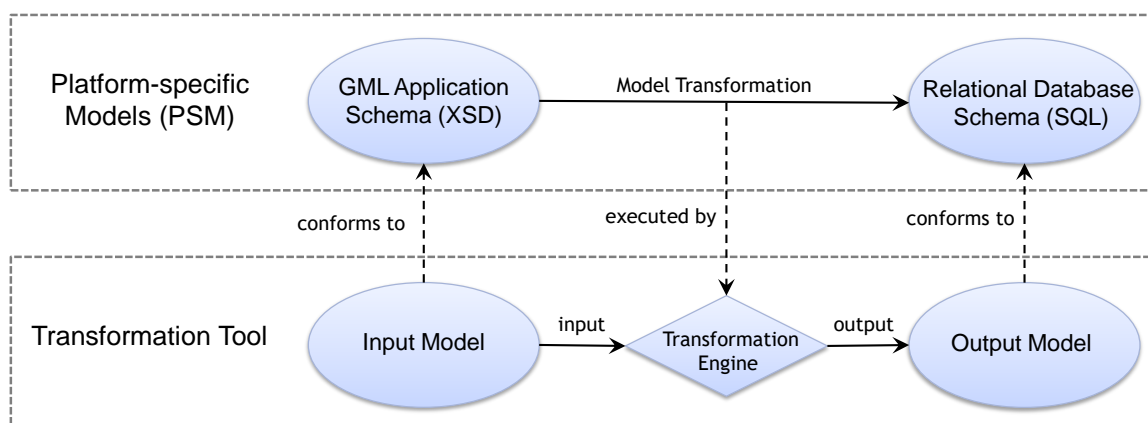


Figure 44: General approach for deriving relational database schema based on model transformation

#### 4.2.1 Survey of existing Transformation Solutions

Today, a number of commercial or open-source software systems have been developed which are capable of reading and parsing GML-conformant application schemas and automatically generating relational database schemas for different relational database management systems. The major ones are listed as the followings:

- Enterprise Architect:** The commercial software “Enterprise Architect” developed by the company Sparx Systems is a visual modelling tool which not only allows users to visually design and develop object-oriented data models in UML diagram, but also to automatically convert the platform-independent data models to domain-specific data models like relational database models (cf. Sparx Systems 2017). If the XML application schema of an ADE is given, the relational database model can be generated using the Enterprise Architect by following two steps: First, the GML application schema shall be read into the Enterprise Architect to reconstruct an object-oriented UML model through a reverse engineering procedure. After this step, a UML package containing all the ADE classes can be created which can be transformed into a relational database model in the second operation step. In addition, Enterprise Architect also provides the functionality to automatically generate the SQL statements for the creation of database schema based on the chosen database products. However,

according to the investigation of this research, Enterprise Architect is not applicable for the handling of CityGML ADEs because of two reasons. First, this software tool can only handle normal XML schema without the support of translating the GML geometric properties to the 2D/3D spatial datatypes in the spatially-enhanced database systems. Second, all ADE classes are mapped onto the database tables in a 1:1 manner which will produce a non-compact database structure and usually result in unsophisticated performance.

- **ShapeChange:** ‘ShapeChange’ is an open-source software tool which has been originally developed by the company *interactive instruments GmbH* and later expanded by the not-for-profit company *MITRE Corporation* (cf. ShapeChange 2017). This software was written in Java and allows for the transformation of geospatial data model between different forms of application schemas according to the ISO 19100 standards. For handling CityGML ADE, ShapeChange is able to directly read the UML model created from the Enterprise Architect and automatically generate the platform-specific formats such as XML application schema and relational database schema etc. Compared to Enterprise Architect, the main advantage of using ShapeChange is that the GML spatial properties can be translated to the spatial data types in the target database systems. In addition, a set of rules and parameters for the fine-grained control of the database derivation process is exposed and can be easily managed by users using an editable XML-based configuration file. For example, a simple conversion rule called ‘*rule-sql-all-associativetables*’ can be activated to ensure that an associative table shall always be created for two classes with an M:N relationship. One of the parameters is named ‘*foreignKeyColumnSuffix*’ which allows users to specify a suffix for the name of those columns that contain foreign keys. In order to choose the database system for which the SQL statements shall be created, the parameter ‘*databaseSystem*’ can be properly specified and the supported values are ‘*PostgreSQL*’ and ‘*Oracle*’ which refer to the two modern database products respectively.
- **Deegree’s Feature Store:** Degree is an open-source web-based application system for the storage, management, and assessment of geospatial data according to the GML specification (cf. Deegree 2017). It is implemented as a Java-tool that allows to handle any GML-compliant 3D city models like INSPRE and CityGML. For storing GML data, Degree provides a system module called *Deegree Feature Store* which uses relational database systems for storing GML data including the spatial properties. The relational database model can be automatically created by reading the GML application schema and mapped onto the database model on top of different relational database products. The outstanding feature of this software is that it provides a comprehensive schema mapping file that can be automatically created and allows to formally describe the mapping information between the GML and database schemas. Using this schema mapping file, the *Deegree Feature Store* can dynamically create SQL queries on the database according to WFS queries to retrieve the data contents from the database via a standardized web-based interface. In addition, the schema mapping file can also be created by the users manually and hence offers a high degree of flexibility to help the advanced developers to create their own database schemas

and express the corresponding schema mapping semantics using a computer-interpretable as well as human readable XML data structure.

- **Snowflake's Go Loader:** Snowflake's *Go Loader* is a commercial data management tool for loading, updating and managing GML-compliant geospatial data in a relational database (cf. Snowflake Software 2016). It provides similar features compared to the *Deegree Feature Store* and also allows to dynamically create relational database schemas for handling arbitrary GML-compliant geospatial data. In addition, its counterpart software tool 'GO Publisher' also enables users to interact with the stored geospatial data with other GIS software applications and to publish the data via open standards like Web Feature Service (WFS). One of the main highlights of the model translation engine coming with this software tool is the capability of decreasing the number of the created tables in the derived relational database schema. It is typically done by mapping the nested complex attributes onto a single table column so that all the corresponding data properties of a feature object will be held in one row of the feature table. This is especially helpful for those GIS applications that are not aware of table joins and can only access one table. Obviously, a positive side effect of such flattening table structure is the fast retrieval of the data contents from the database as many table joins can be avoided. However, fine-grained data queries are restricted because the database is not able to identify the individual attributes of its parent complex attribute stored in a table cell.

After the survey of the existing model transformation solutions, it is identified that most of them are capable of deriving valid relational database schemas for any GML-compliant ADE data but are very limited in creating a compact relational structure, because the underlying mapping rules for the model transformation are too simple. For instance, each GML class or complex data type is normally mapped onto one individual database table, and the association between two GML classes is represented using a table join or an associative database table to connect the two mapped database tables. Such kind of mapping rule can easily result in a large amount of database tables and will cause time-consuming issues when, for example, performing a complex query on those data contents that are distributed over many database tables due to the large number of required database joins. Although this issue has been taken into account by the software "Go Loader" which allows for mapping a class along with its complex attributes onto one database table to ensure that the overall number of the generated database tables can be minimized. However, such optimization rules are hard-coded within the software and lacks the flexibility to support user-defined transformation rules having complex logics. For example, a user may want to map two classes onto one table rather than two separate ones, in case that the two classes have many common attributes or properties being inherited from a common parent class, because such table representation is not only space-efficient for database but also has much more compact database structure to speed up the database access for GIS applications like ETL software tools. Such kind of mapping rules allows to achieve a very good trade-off between database complexity and semantic clarity and can be learned from the design decisions applied during the development of the 3DCityDB relational database schema. Thus, it would be sufficient to find a way to formally represent the 3DCityDB's mapping rules that can be interpreted by computer to automate the database derivation process for arbitrary CityGML ADEs. However, the logics of the 3DCityDB's

design decision is relatively complex and make it difficult to reuse or extend the existing solutions being used in the aforementioned software tools due to the following reasons:

- According to the literature (Stadler et al. 2009), the manual derivation of the 3DCityDB mainly consists of two mapping steps: First, the classes matching to some certain object-oriented design patterns are identified and optimized to a simple model structure with less classes and relations. In the second step, the simplified object-oriented model shall be transformed to the target relational database model in a straightforward manner using the standard mapping approaches (cf. chapter 3.2.1). However, particular problems occur when trying to automate the first mapping step using the solutions of the traditional software tools, because the class patterns may have varying characteristics and complex structures which can easily result in a situation that a class matches multiple patterns of different mapping rules at the same time. This is a typical conflicting issue that will confuse the software tool, which mapping rule should be chosen for performing the model transformation on the classes.
- For specific cases, users may want to add further mapping rules. However, the existing software solutions do not support the dynamic enrichment of user-defined mapping rules to the transformation engine, because the mapping rules are normally hard-coded in the application systems using high-level programming language or annotated in the original XML application schema (cf. Amer-Yahia et al. 2004). Thus, a flexible and declarative formalism is required which should allow users to formally express the mapping rules that can be automatically executed by the computers to perform the model transformation. In addition, the mapping relationships between the object-oriented model and the relational database model shall also be represented to reflect, which classes or attributes are mapped onto which tables or columns. Moreover, the relational model derived from the mapping process shall also be represented properly such that it can be interpreted by computers to automatically generate the target database schema.
- With the increasing number of user-defined mapping rules, the mapping process can become even more complicated. One typical issue is that the transformation results of a mapping rules may fulfill the condition of another transformation rule, whose transformation results may in turn match the mapping condition of the former mapping rule. This will result in an infinite loop of the transformation process which cannot be terminated in the end. Therefore, an appropriate mechanism is required to be tailored to such situations which can be automatically determined beforehand and allow users to schedule the sequence of running the individual mapping rules to ensure that one mapping rule can only be started when another one has been completely processed.

Concerning these issues, a new approach must be found which should be able to overcome the above-mentioned drawbacks of the existing software solutions to support the automatic derivation of a compact relational database model form a give CityGML ADE. After careful examination of the object-oriented model and relational database model, it can be concluded that both model representations can be fully represented using a graph structure along with its typed and attributed graph nodes and arcs which can represent the model entities and their



semantic interrelationships respectively (cf. Kuske et al. 2009). In addition, the mapping relationships between both model objects e.g., classes and database table objects can also be properly expressed using directed graph arcs which connect the corresponding graph nodes. Thus, the model transformation problem can be fully abstracted to a graph transformation problem which can be solved in a systematic way by means of so-called *Graph Transformation Systems* (cf. Taentzer et al. 2005).

#### 4.2.2 Relevant Concepts of Graph Transformation System

Graph Transformation System (GTS) is a very important technique in the area of computer science and has been widely used in various applications ranging from software engineering up to computation simulations for different algorithms, data and model transformation (cf. Vara et al. 2005, Rafe et al. 2011). The fundamental concept of using Graph Transformation Systems is the algebraic approach which uses graphs to represent the computation logics for solving some specific problems. According to the graph theory, a graph  $G$  is commonly defined as  $G = (V, E)$  where  $V$  represents a set of nodes, whereas  $E$  denotes the edges. Each edge must be bounded by two nodes which can be identical or different. A node can be connected with one or more edges, the number of which indicates the *degree* of the respective node (cf. Godsil & Royle 2013). In graph transformation system, the graph definition has been extended to a special kind of graph whose nodes and edges are labeled with different types and attributed with a set of characteristics. Such graph is commonly called “*typed attributed graph*” and can be seen as a suitable grammar for formally describing a various kinds of computation states and activities of human behaviors, object movements, software engineering processes etc. at a higher abstract level. For example, the UML diagrams such as component diagram, activity diagram, as well as the class diagrams can be fully mapped onto a typed attributed graph which allows for representing the semantics of the target applications and data structures within a graph transformation system (cf. Büttner & Gogolla 2004, Folliet & Mens 2008).

When using a graph transformation system, the initial state of the computation process is formulated as a typed attributed graph called “host graph” which shall be first created and will be used as the input for the graph transformation process. The entire process is carried out by means of a set of user-defined transformation rules called ‘graph transformation rules’ each of which uses graph representations to formalize the logical operation or computations on the host graph and will be applied to the host graph to replace a certain number of subgraphs by a new subgraph iteratively (cf. Ehrig et al. 2015). The computation results will be yielded by interpreting the processed host graph once the graph transformation has been completed (cf. Figure 45). In addition, the graph transformation rules can also be flexibly combined to perform complex computations. A nice application example was given by (Krüger & Kolbe 2010) who have developed a GTS-based framework for the analysis, interpretation, and transformation of 3D geospatial data according to the GML specification. For example, the data transformations on the feature-based datasets allow for the data generalization by merging multiple objects by analyzing their object attributes, the topological, and geometrical relations. Since the GML data has a graph-like structure, the source data including the feature objects and their attribute properties can be directly mapped onto a typed attributed graph and

the data transformations can be declaratively defined as graph transformation rules for performing the transformation in a graph transformation system.

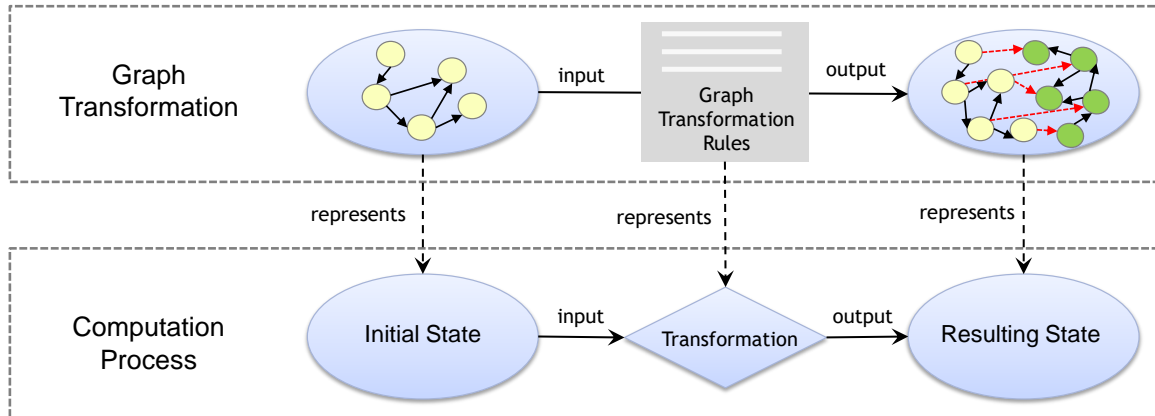


Figure 45: Conceptual workflow of using graph transformation system to perform a computation process

Formally speaking, the graph transformation rules can be denoted as  $\{r_1, r_2, \dots, r_n\}$  each of which is equivalent to a match morphism which basically consists of two components and can be generally formalized as  $r: L \rightarrow R$  where  $L$  is called left-hand side (LHS) graph, whereas  $R$  is called right-hand side (RHS) graph (cf. Figure 46). Both LHS and RHS are also typed attributed graphs. The LHS graph can be considered as a match pattern which could be algebraically isomorphic to every member of those graphs  $\{G'_1, G'_2, \dots, G'_n\}$  that are subsets of the host graph  $G_S$ , where  $L \cong G'_x$  and  $\{G'_1, G'_2, \dots, G'_n\} \subseteq G_S$ . When executing a graph transformation, each of the given rules will be checked and if a match  $m(L)$  of the respective LHS has been found in the host graph, the respective graph transformation rule will be considered to be applicable and the matched subgraph in the host graph  $G_L$  will be substituted by the RHS. Subsequently, the modified host graph  $G_R$  will in turn be treated as the input for the next transformation step and the entire graph transformation process will be continued by successively processing the transformation rules until there is no further match of their LHS can be found in the host graph.

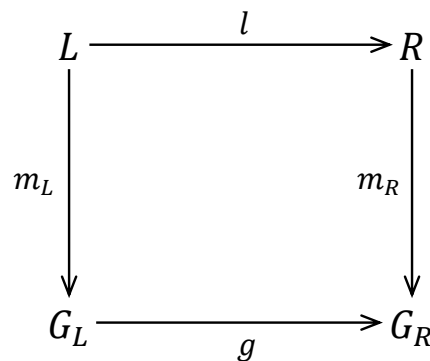


Figure 46: Conceptual diagram of the SPO-based approach (cf. Geiß et al. 2006)

The upper-mentioned approach is generally called single-pushout approach (SPO) which reflects the basic idea of graph transformation. With this approach, a graph transformation rule with matched LHS can be directly applied to the host graph without the need to fulfil any further conditions for running the graph transformation process. However, such straightforward rewriting process may easily result in a “dangling” edge (cf. the example

shown in Figure 47), which does not have a source or target node and violates the fundamental definition of a graph.

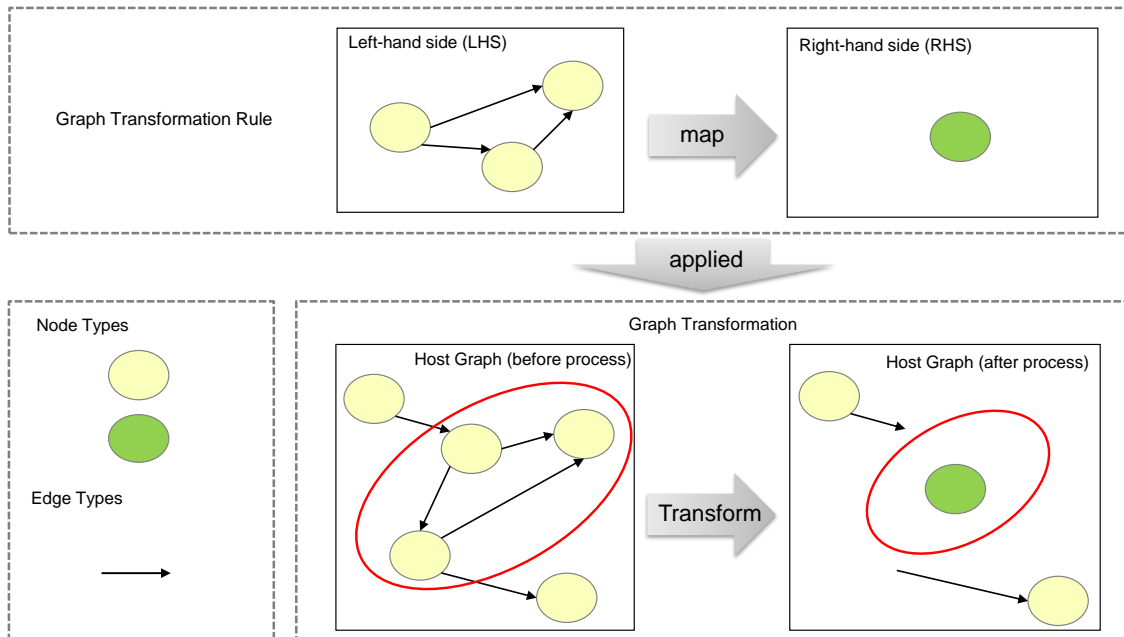


Figure 47: Example of the “dangling” edges resulted from a simple graph rewriting process

As shown in Figure 47, a simple transformation rule has been defined which tends to map a graph structure consisting of three interrelated nodes and edges to a simple graph containing only one node with another type. After applying the transformation rule to the host graph, the matched subgraph will be substituted by the RHS-graph of this transformation rule and the resulted graph shown on the right side of the figure will contain two dangling edges. As mentioned above, the resulted graph has an invalid structure and must hence be automatically repaired through the automatic deletion of the two dangling edges (cf. van den Broek 1991). Here, a considerable issue of the SPO-based approach is that the deletion of edges will be instinctively performed by the graph transformation system and cannot be fully controlled by the graph transformation rules.

An alternative to the SPO-based approach is called double-pushout (DPO) approach which is a much more rigorous approach having been frequently used for solving the graph transformation problems. The modern graph transformation system AGG (cf. Taentzer 2000) is implemented based on the DPO-based approach at their core. According to the concept of this approach, a graph transformation rule can be described as  $r: L \leftarrow I \rightarrow R$  which is slightly different from the formalization for the SPO-approach due to the additional variant  $I$ . It represents a so-called gluing graph ( $L \cap R$ ) which is equivalent to an interface for joining the both LHS-graph and RHS-graph. The two morphisms  $I \rightarrow L$  and  $I \rightarrow R$  are said to be injective since the graph  $I$  is actually a subgraph of the both  $L$  and  $R$  (cf. Ehrig et al. 2006).

For performing a DPO-based transformation, a graph transformation rule is applicable only if it satisfies the so-called “gluing condition” which is made up of two sub-conditions, namely the *identification condition* and the *dangling condition*. The identification condition restricts that, if two graph elements (node or edge)  $x$  and  $y$  of the LHS-graph are not going to be deleted by the rule, then they must occur in the gluing (interface) graph. In other words, the

two elements will be preserved after applying the transformation rule if  $x, y \in L \cap R$ . The dangling condition requires that if one node is to be deleted, then all edges which are adjacent to this node must also be deleted. This condition is particularly important since it allows to avoid the occurrence of dangling edges in the output graph to preserve a legal graph structure. Once the gluing condition is satisfied, the DPO-based transformation can be executed by performing a two-steps procedure which are formally illustrated in Figure 48.

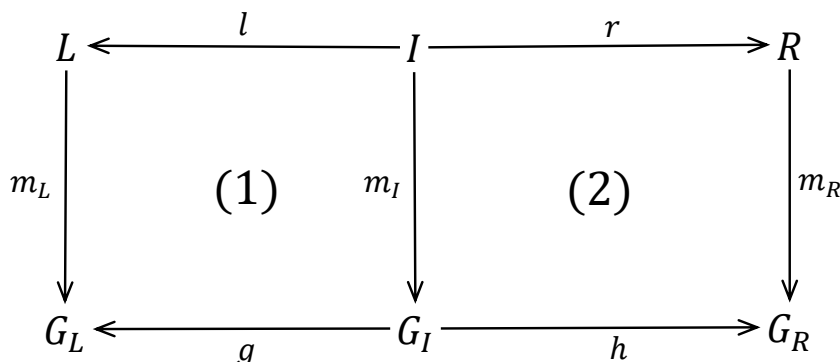


Figure 48: Conceptual diagram of the DPO-based approach (cf. Habel et al. 2001)

In the first step, all those graph elements that match the subgraph  $L \setminus I$  shall be deleted from the host graph  $G_L$ . An intermediate graph  $G_I$ , which is production of the  $G_L \setminus m(L \setminus I)$ , will be yielded which contains a match  $m(I)$  of the gluing graph  $I$ . According to the dangling condition, no dangling edges will be created in the graph  $G_I$  during the transformation. In the second step, the intermediate graph  $G_I$  will be attached with the production of  $R \setminus I$  to generate the result graph  $G_R$ . Since the gluing graph  $I$  occurs in both LHS-graph and RHS-graph, the input  $G_L$  and output  $G_H$  of the host graph can share a common subgraph that allows to connect the initial and final computation states represented by the graph (cf. Habel et al. 2001). In case that none of the original graph elements have been deleted during the rule applications, the structure of the original graph can be completely preserved. This feature is especially essential for performing model transformations, because not only the target model but also the mapping relationship between the input and output models can be kept which are important for exploring the transformation results.

For the clear understanding, a simple example is given in Figure 49 to illustrate the general workflow of the DPO-based approach. The transformation rule shown in the example tries to convert a graph pattern by adding a new node along with three adjacent red edges linking with the original nodes. All graph elements in the host graph remain unchanged throughout the transformation process and the rewritten host graph can be treated as a graph-based representation of the computation results which preserves the input graph elements and also contains the relations between the initial and newly created graph elements. A practical circumstance of this example is model transformation: The yellow nodes can be seen as the virtual representation of an object-oriented model consisting of three classes whose inheritance or association relationships can be represented using the black edges. The newly created green node represents a database table which is logically mapped from the three given classes and the related red edges can be used for holding the mapping relationships between the respective model objects. In this way, the mapping of complex-structured class patterns onto a relational database model can be easily formalized as a graph transformation rule that

can be applied using the graph transformation system to accomplish the complex model transformation task.

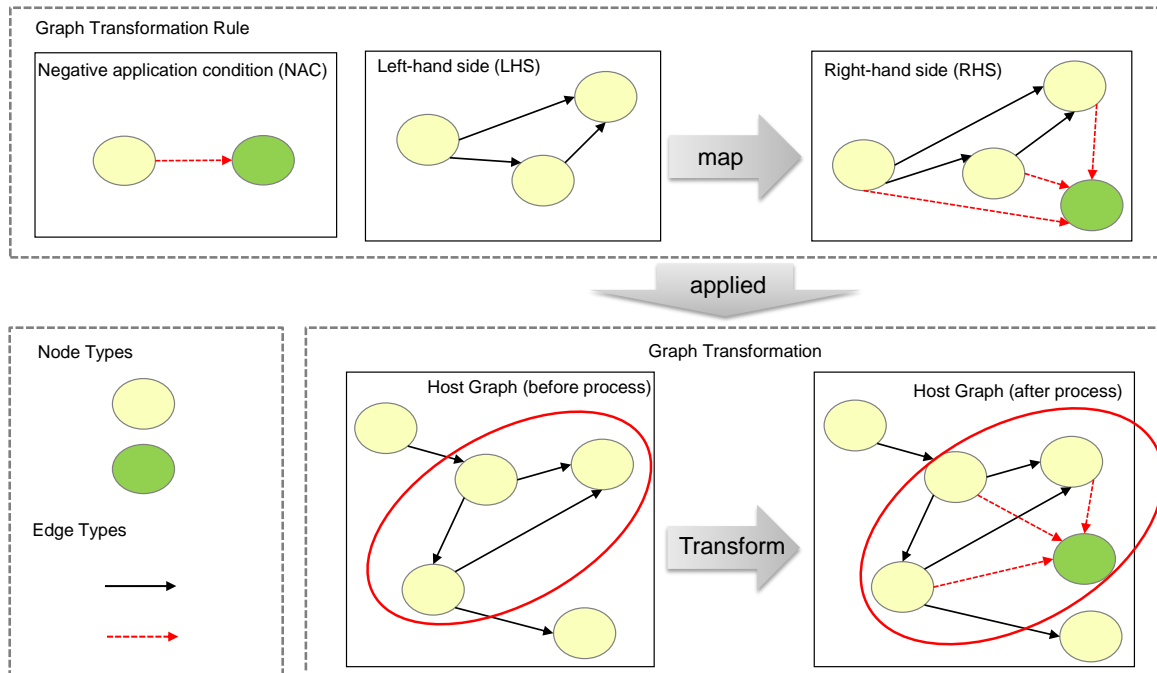


Figure 49: Example of DPO approach for graph transformation

An issue in the upper example is that, the graph transformation process can actually not be terminated, because the LHS always has a match in the host graph and the respective transformation rule will be applied on the host graph in infinite times (cf. Bottoni et al. 2005). Hence, the applicability of every graph transformation rule must be further restricted to prevent it from running multiple times. This can be reached by augmenting a transformation rule with a set number of pre-conditions which are typically categorized into two types: namely the *Positive Application Condition* (PAC) and the *Negative Application Condition* (NAC), both of which are also expressed in graph structure. The PACs are combined with the logical operator “AND” so that a graph transformation rule will only be triggered when all its PACs are fulfilled. In the contrary, the NACs use the logical operator “OR” and make a transformation rule not to be applicable if one of the NACs is satisfied. Therefore, the proper use of PACs and NACs offers a variety of possibilities for specifying a graph transformation rule being subject to different kinds of application conditions. In the upper example, a NAC has been introduced to restrict that if one yellow node has been already connected with a green node, then the respective transformation rule will become non-applicable. In this way, the transformation rule can be only performed one time and the graph transformation process can therefore be properly terminated when only new graph nodes and edges are created in the host graph. This corresponds to the scenario of deriving relational database models from object-oriented models.

Additionally, it is also possible to define a so-called *type graph*  $T$  (cf. Figure 50) which is an essential concept in graph transformation systems. It can be seen as a meta-graph over all graph elements for prescribing their structural relationships in the host graph as well as in all the defined graph transformation rules. For instance, a semantic relation between two node types can be defined in a meta-graph to represent their inheritance relationship. This allows

that, if a transformation rule contains the super node type, it will also be applicable for those matched subgraphs that contain the child node type, since the semantics of the parent node type have been inherited by its subtype. In addition, an edge type can also be restricted by defining the multiplicity of its associated source and target nodes. For instance, in the upper example, if the red edge type in the meta-graph has been defined in such a way that, the edge can only link a yellow node with maximum one green node, then the graph transformation can also be properly terminated without needing to use a NAC. This indicates that the use of the meta-graph has very similar functionalities compared to the UML meta-model being used in the model-driven engineering. Thus, the meta-graph is able to serve as a global constraint on all the graph elements in the graph transformation system to guarantee the structural consistency of the host graph and transformation rules throughout the transformation process.

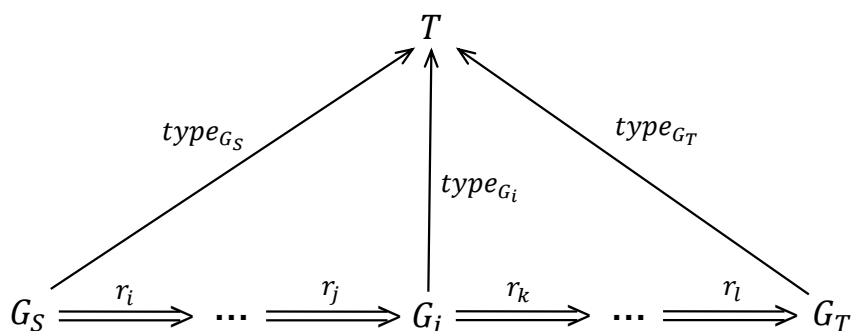


Figure 50: Conceptual diagram of graph transformation with type graph (cf. Taentzer et al. 2006)

While applying a large number of graph transformation rules, non-determinism or also called “conflicting” can easily occur, if more multiple rules are applicable to the host graph at the same time (cf. AGG 2006). In this case, one of these matched rules might be randomly chosen to be first applied which may affect the other rules becoming not to be applicable to the rewritten graph. For example, one rule application may tend to delete a graph node or edge which simultaneously has a match in the LHS-graph of another rule. Another typical example is that a graph object created by a rule application may occur in a NAC-graph of another rule. The negative consequence of these situations is that, non-confluent output graph or even non-terminated transformation can easily occur which will result in incorrect application results. Thus, such kinds of dependencies between transformation rules must be avoided in a proper way.

To solve this issue, graph transformation systems like AGG provide a very powerful control-flow mechanism called layered-based transformation which is an important concept for handling the conflicted application rules (cf. Ehrig et al. 2005). The general idea is to schedule the processing sequence of all given application rules by grouping them into a set of numbered layers which can be sorted in a descending order based on the user-defined execution priorities and as such will be applied successively. With such layer subdivision, each layer will have a small number of application rules whose potential conflicts can be easily determined and eventually avoided by decomposing the conflicted rules into further layers. In addition, each layer shall be applied only one time while performing the graph transformation and may conform to a logic block for accomplishing a certain sub-task of the entire processing-chain. In each layer, the included application rules shall be applied as long as possible and the processed host graph will be delivered as the input for the next layer until

the processing of the last one has been successfully completed. Thus, a proper ordering of the layers is of great importance for achieving the desired transformation result and hence must be carefully designed by application developers. The conceptual workflow of the layered-based transformation is shown in Figure 51.

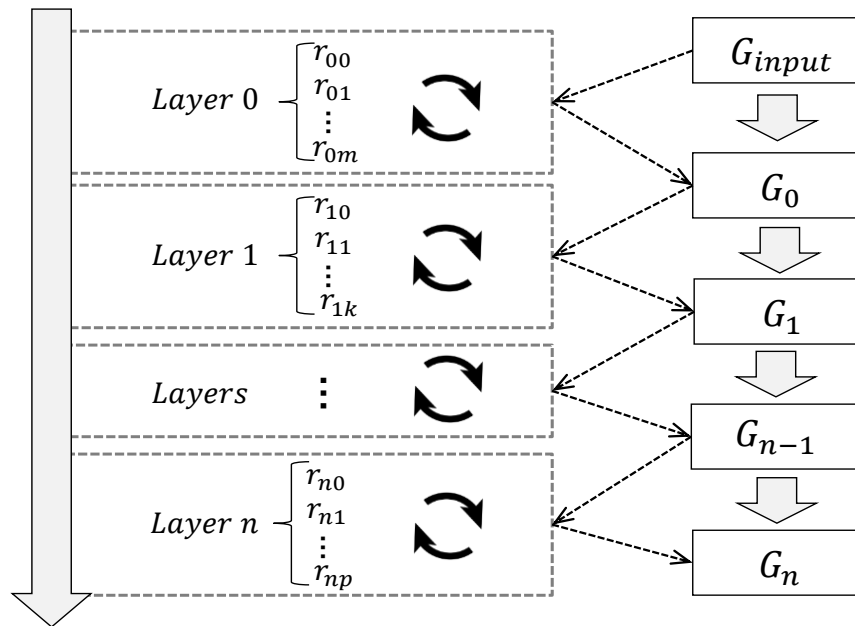


Figure 51: Conceptual workflow of layer-based graph transformation

#### 4.2.3 Concept of using GTS to automatically derive ADE Database Schemas

Based on to the presented concepts of the graph transformation system in the previous subsection, a new graph-based approach has been investigated in the context of the thesis to develop an advanced application system for realizing the automatic derivation of a compact as well as 3DCityDB-compliant relational database schema for a given CityGML ADE (cf. Yao & Kolbe 2017). The conceptual workflow of the developed approach is mainly made up of the following four steps:

1. In the first step, the XML schema definition (XSD) file of the input CityGML ADE is read into the application for parsing the contents of the underlying object-oriented data model which will be subsequently mapped onto a graph structure. The contained classes along with their attributes and associations are represented as a set of typed attributed graph objects which allow to fully reflect the semantics of the respective ADE data model. In addition, by using a meta-graph, the meta-model of the CityGML ADE can also be expressed in the graph transformation system to guarantee that the graph representation of the ADE data models always conform to the GML specification.
2. In the second step, the mapped graph will be passed to the graph transformation system where a number of graph transformation rules can be defined to formulate the complex mapping rules learned from the 3DCityDB implementation for performing the transformation from the object-oriented model to the relational database model. For this step, modern graph transformation systems usually provide an intuitive graphical editor allowing users to conveniently define the transformation rules by

dragging and dropping the graph elements. Moreover, the mapping relationships between both models are explicitly represented using a number of graph edges with a specific type (painted with red color in Figure 52) for connecting the source and target model objects.

3. In the third step, the resulted graph shall be parsed and the subgraph representing the derived relational database model will be retrieved by navigating the respective graph objects. This can be done by programmatically iterating through all those nodes and edges that are typed for representing the relational database objects such as database tables, columns, sequences, indexes, and foreign key constraints etc. Starting from a table node, its ingoing and outgoing edges can be queried to rebuild the relational database structure which can be directly translated to the corresponding SQL statements for the creation of the relational database schema according to the chosen database product.
4. In the last step, the mapping relationships between the GML application schema and the its relational database schema shall also be parsed and formally represented using a text-based format which can be easily transmitted as well as interpreted by human operators and computer applications. For this purpose, an XML-based file format has been developed which is an extended version of the Deegree schema mapping file for supporting the description of the database mapping employed in the 3DCityDB implementations. Using this schema mapping file, the XPath expression can be easily translated to the corresponding SQL query statement for retrieving the data contents from the database and hence allows to build a WFS-based application system (cf. Almendros-Jiménez et al. 2008). In addition, some meta-information of the CityGML ADE can also be carried in the schema mapping file which can be used for the registration of the respective ADE into the 3DCityDB database.

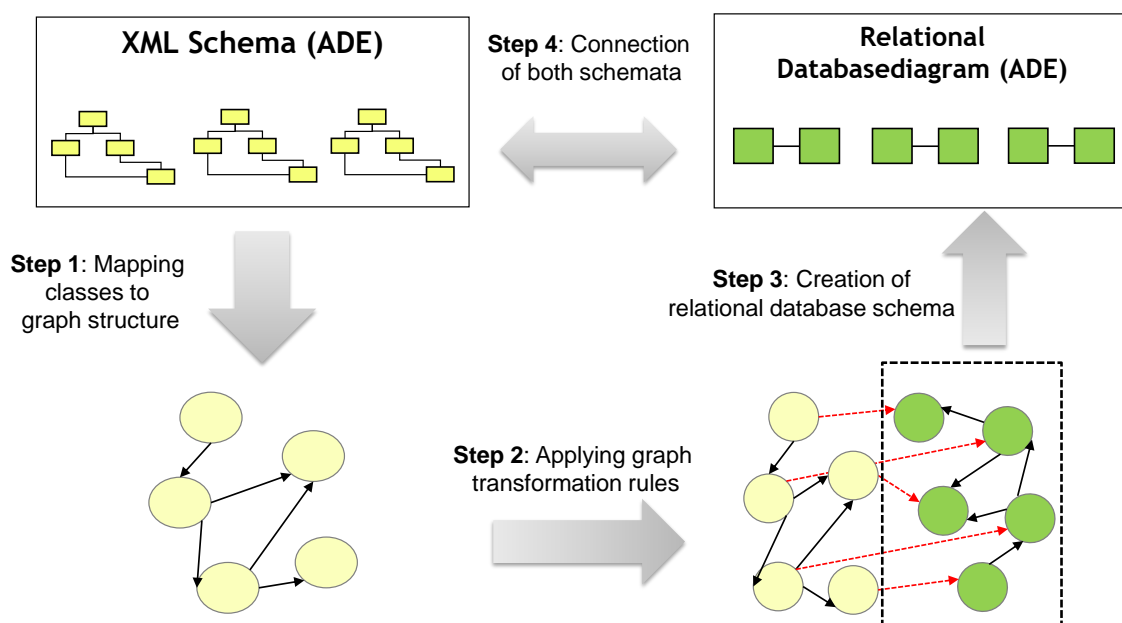


Figure 52: General idea of deriving a relational database schema from a GML application schema by means of graph transformation



### 4.3 Implementation and Evaluation

Based on the conceptual solution outlined in the previous section, a prototypical converter tool (cf. Figure 53) for performing the database derivation has been implemented which is a stand-alone Java application on the basis of the AGG graph transformation system (cf. AGG 2006). It allows to read the ADE application schema using the Java library XSOM for parsing the XML elements of the data models and map them onto an AGG-compliant graph representation, which can be directly passed to the AGG's graph transformation engine for executing the model transformation. In addition, the converter tool also includes the functionalities for parsing the AGG host graph to automatically generate the SQL-formatted relational database schemas and the corresponding schema mapping file once the graph transformation has been completed. The applied graph transformation rules as well as the meta-graph have been predefined using the graphical editor provided by AGG and integrated into the developed converter tool. Moreover, a new database structure has also been designed for the 3DCityDB which can now be dynamically extended to cope with an arbitrary number of ADEs.

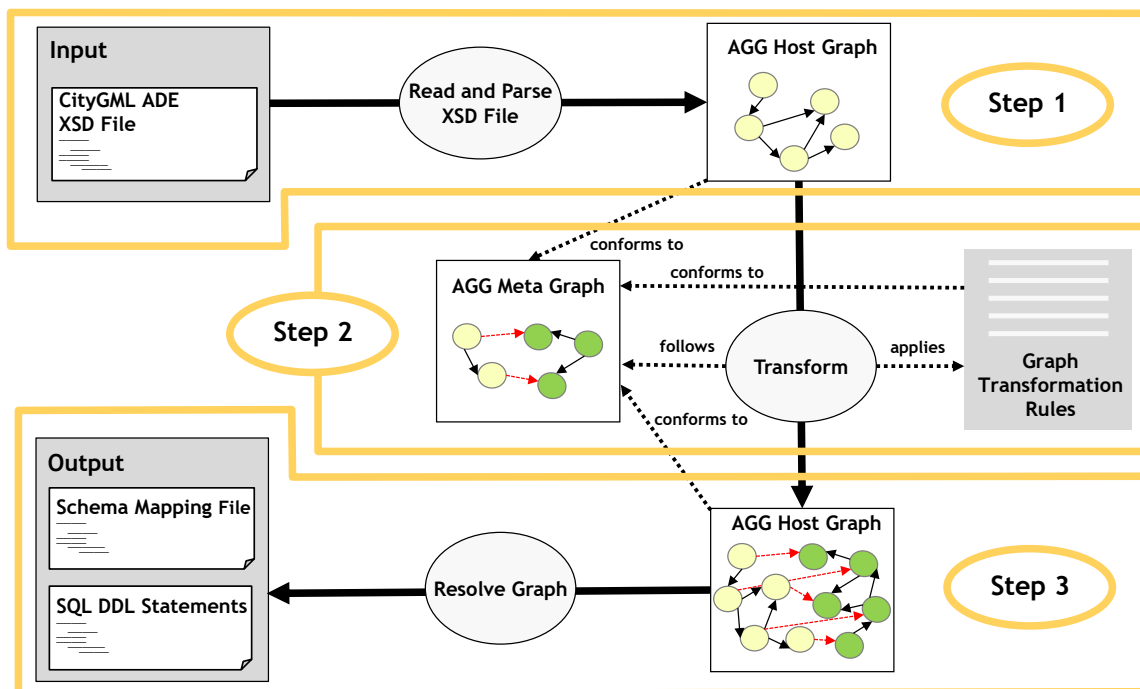


Figure 53: Software structure of the developed graph-based converter tool for generating relational database schema and schema mapping file from a given ADE application schema

In the following subsections, the main design decisions for the graph transformation environment including the meta-graph and the graph transformation rules will be first presented to show how to create a graph-based as well as model-driven like system for realizing the derivation of an efficient relational database schemas from a given CityGML ADE. In the second subsection, the new structure of the 3DCityDB database schema will be introduced which allows users to easily manage the derived ADE database schemas within the 3DCityDB software environment. In the last subsection, the developed graph-based approach will be evaluated by comparing it with the existing software systems like Enterprise Architect, Degree, and ShapeChange based on their generated relational database schemas of an

‘artificial’ ADE. This artificial ADE has been specifically designed for simulating the typical data model structures which frequently occur in practical application scenarios.

### 4.3.1 Design of a Graph Transformation Environment

Since a correct meta-graph can guarantee the validity of the host graph throughout the graph transformation process, it is important to carefully design the meta-graph for both object-oriented and relational database models. The structure of the meta-graph can first be conceptually expressed as a UML diagram and later be manually mapped onto the corresponding graph structure, which can be interpreted in graph transformation systems. Basically, the meta-graph is made up of two subgraphs which are used for representing the GML application schema model and relational database model respectively. In Figure 54, the UML diagram of the subgraph representing the meta-model of the GML application schema is presented.

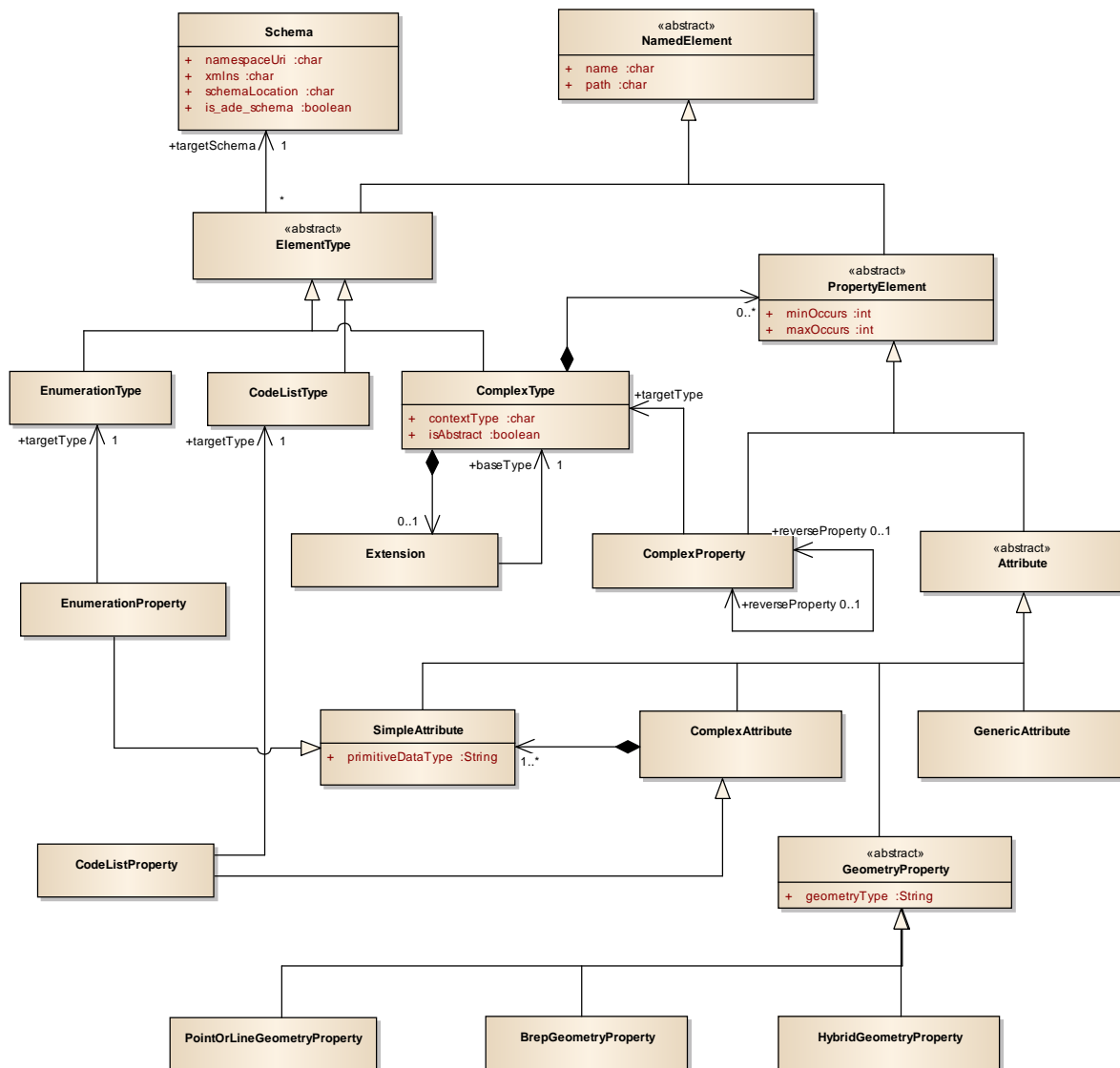


Figure 54: Meta-model of the GML application schema according to the ISO 19136

This UML diagram was manually created via a reverse engineering of the GML conceptual meta-model according to the ISO 19136 standard. In this UML diagram, all classes and their

relations are abstracted from the XML encoding rules specified in the ISO 19136 Annex E. First, an abstract class *NamedElement* is given which is the top-most class for representing the XML elements and types whose names are held using the textual attribute *name*. This class can be further specialized into two abstract classes namely *ElementType* and *propertyElement*. The *ElementType* along with its three subtypes (*EnumerationType*, *CodeListType* and *ComplexType*) corresponds to the global XML *complexType* and *simpleType* elements which are specifically encoded for representing those GML classes that are defined using the UML stereotypes like `<<Enumeration>>`, `<<CodeList>>`, `<<DataType>>`, `<<Type>>`, and `<<FeatureType>>` (cf. chapter 2.4). Unlike the *EnumerationType* and *CodeListType* the *ComplexType* class is able to contain an arbitrary number of property elements which are represented using the class *PropertyElement* and the multiplicity information can be stored using the *minOccurs* and *maxOccurs* attributes.

The *ComplexType* is also a simplified class for representing the stereotype `<<DataType>>`, `<<FeatureType>>`, and `<<Type>>` simultaneously. The reason of employing this simplification is that the three class types have a common encoding structure regarding the inheritance relationships and associations. For example, the inheritance relationship is represented by means of an *Extension* element to connect the parent and child class according to the extension mechanism of XML schema. Besides, the association relationship like aggregation and composition between two classes is mapped onto an XML property element which is represented using the class *ComplexProperty* by following the so-called “*complexType-property-complexType*” encoding structure where a super class can contain an element property which shall have a reference to the child class to build the relation. Note that only directed associations are allowed when two classes are connected through a property element. In order to represent a binary association, the child class needs to receive another property element pointing the parent class and both property elements shall be paired using the *reverseProperty* attribute.

Another subtype of the *PropertyElement* class is called *Attribute* which is mainly used for representing the non-association property elements. It can be further specialized into four classes. The first one is *SimpleAttribute* which represents primitive attributes like integer, string, floating-point number, boolean, and date etc. The *EnumerationProperty* is a special case of the *simpleAttribute* since the value defined in the *Enumeration* is always a simple textual value according to the definition in the ISO 19136 standard. The *ComplexAttribute* corresponds to those kind of GML data type that is an aggregation of multiple simple attributes. For example, the GML *MeasureType* consists of two simple attributes which store the value and unit information respectively. A special subtype of the *Attribute* is the *GeometryProperty* which is specifically used for representing the GML-compliant geometry properties and is further classified into three concrete types, namely *PointOrLineGeometryProperty*, *BrepGeometryProperty*, and *HybridGeometryProperty*. As the class names imply, the *PointOrLineGeometryProperty* represents those geometry objects that are constructed using 3D points or curves. In case of B-Rep-based geometries 3D surfaces and solids, the *BrepGeometryProperty* shall be employed. If a geometry object is an aggregation of different geometry types, the *HybridGeometryProperty* shall be used. Since an ADE schema may also contain some data property types which are not conform to the ISO 19136 standard, such kind of property elements can be represented using the *GenericAttribute*

class which will be mapped onto a single table column in the database using the data type BLOB for storing the respective XML data fragment.

The UML model of the subgraph representing the relational database schemas is illustrated in Figure 55. Different kinds of database objects such as the database tables, indexes, columns, sequences, as well as foreign key constraints are mapped onto the individual classes and their relations are also reflected.

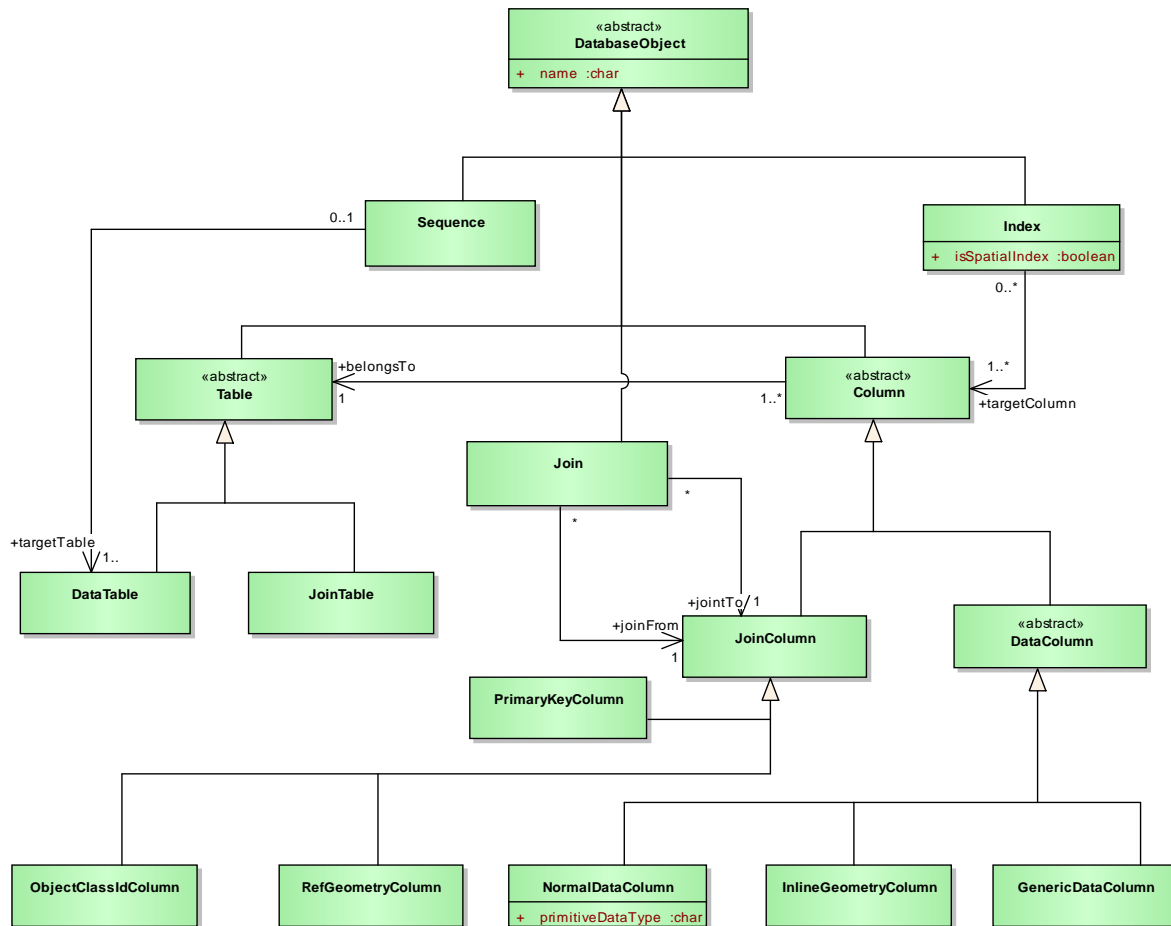


Figure 55: Meta-model of the relational database model

In this meta-model, an abstract class called *DatabaseObject* is the top-most class for holding all the relevant types of database objects. As a subtype of it, the *Table* class is used for representing the database table which normally contains multiple columns represented using the *Column* class. According to the usage functionality, the *Table* class can be further specialized into two classes: *DataTable* and *JoinTable*. The *DataTable* refers to those tables that are utilized for storing the data contents of the object classes and their attributes, whereas the *JoinTable* exclusively serves as an associative table to link the tables of two classes having an M:N relationship. For each data table, a sequence object can be defined which allows to automatically generate a set of incrementing and unique values which can be propagated to the primary key column of the respective table. Thus, the *Sequence* class and the *DataTable* class shall have a 1:1 relationship. The database index being used for speeding up the data accessing performance is represented using the class *Index*. It can be either a normal index for indexing the primitive data types or a spatial index for indexing the spatial data types. Both index types are determined using the flag attribute *isSpatialIndex*. Since a

normal index is able to be defined on multiple columns of a table, and more than one index can also be defined on the same column, the classes *Index* and *Column* are associated with an 1:N multiplicity.

The Class *Column* has two subtypes *JoinColumn* and *DataColumn*, the latter of which can be further classified into a number of subclasses. For example, the class *NormalDataColumn* corresponds to those columns that are used for storing the attributes defined with primitive data. The *InlineGeometryColumn* corresponds to the spatial data columns supported by the spatially-enhanced relational database management systems. In addition, the *GenericDataColumn* is a specific column with the database's data type CLOB for holding the complex-structured XML data elements. The *JoinColumn* class is a numeric column which can be used for linking and joining two tables and can hence be treated as one table's foreign key column referencing to a primary column of another table. Such logical join relationship is represented using the *Join* class which can be implemented as a foreign key constraint allowing to guarantee database integrity and consistency. The referencing direction pointing from a foreign key column to the target *primaryKeyColumn* can be realized using the *joinFrom* and *joinTo* associations. Since a foreign key can also be defined on a primary key column, the *PrimaryKeyColumn* class is modeled as a subtype of the *JoinColumn* class. In addition, there are two specific join columns namely *ObjectClassIdColumn* and *RefGeometryColumn* both of which are exclusively used as foreign key columns referencing to two existing 3DCityDB tables. The *ObjectClassIdColumn* is a foreign key of the 3DCityDB table OBJECTCLASS and is typically used for determining the class affiliation if a data table is mapped from multiple classes. The *RefGeometryColumn* references to the 3DCityDB table SURFACE\_GEOMETRY for efficiently storing the B-Rep-based geometry properties.

To illustrate the basic concept of the mapping approach, a subset of the schema mapping model has been chosen for the discussion and presented in Figure 56.

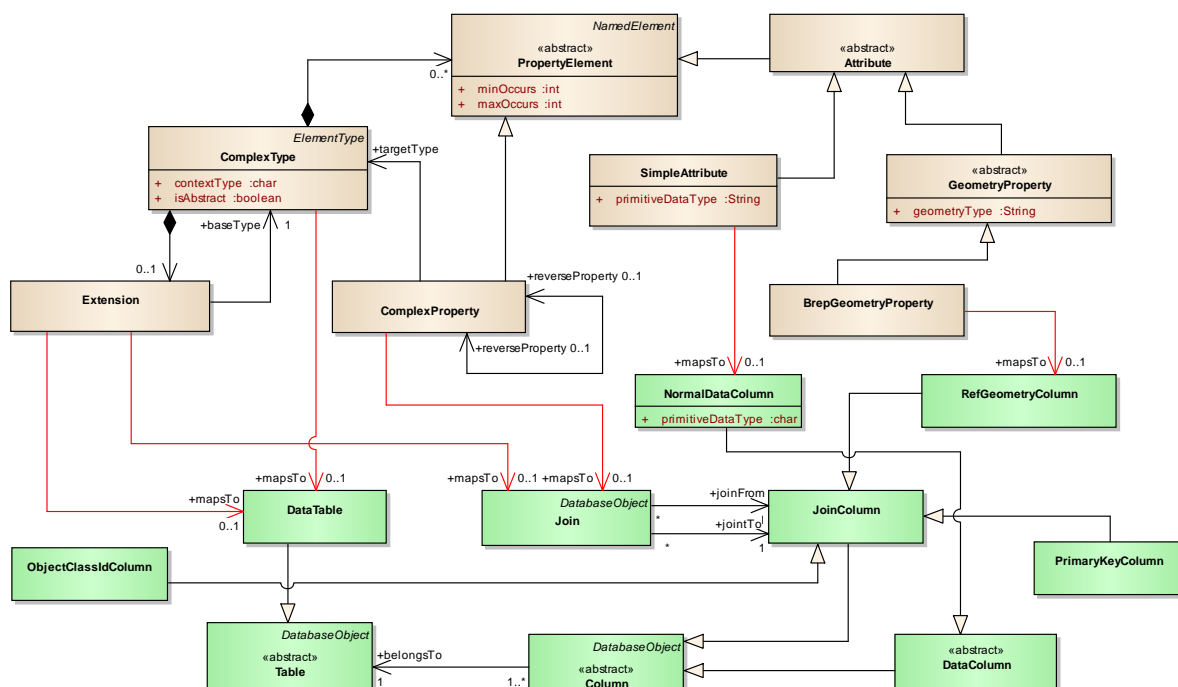


Figure 56: An excerpt of the meta-graph for representing the model mapping structure

In the UML diagram (cf. Figure 56), the conceptual mapping relationships between the ADE's GML application schema and the relational database schema are realized using a set of directed associations to link the individual model objects. For example, the mapping from classes onto tables has been designed according to the following concepts:

- A feature class shall be mapped onto one table, where each row should store the attribute information of a feature instance. It is allowed to map multiple ADE feature classes onto the same table only in the case that the classes belong to the same inheritance hierarchy.
- If two feature classes with inheritance relationship have been mapped onto two separate tables, a foreign key constraint shall be used to hold the inheritance relation in databases. In another case where the two ADE classes are mapped onto one table, the inheritance relationship itself should be logically mapped onto the same table at the same time.
- In case that two feature classes are not directly or transitively inherited, they have to be strictly mapped onto two separate tables. If the two classes are associated with a 1:N or N:0..1 multiplicity, the two mapped tables shall be joined using a foreign key constraint which will be logically used for representing the mapping of the association relationship.

Above, a variety of mapping rules can be declaratively expressed using graph transformation rules, which can be combined to realize more complex mappings e.g. the handling of the Composite design pattern. The first transformation rule is the mapping of a single feature class onto a table which shall at the same time receive a primary key column to uniquely determine the instance objects within the table. The corresponding graph transformation rule is shown in Figure 57, which was drawn using the graphical editor of the AGG graph transformation system. Note that since the meta-model (cf. Figure 56) has already specified that one class shall be mapped onto maximum one table, there is hence no need to define a NAC for preventing this transformation rule from running multiple times on the same class.

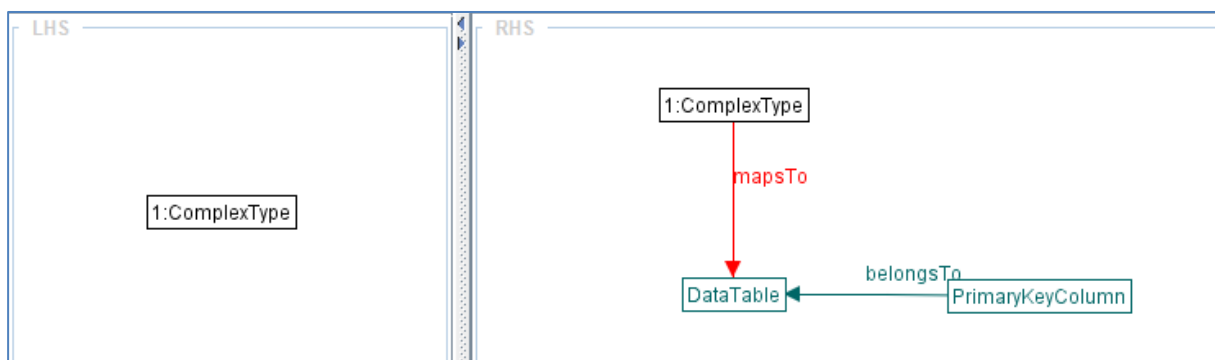


Figure 57: Rule 1: Mapping ADE class onto table

Once every class has been mapped onto a table, the next mapping rule is needed for handling the inheritance relationships among classes. Depending on their relations and attribute properties, two classes can be mapped onto one table to generate a compact relational structure which allows not only to decrease the number of table joins for speeding up the data query performance, but also to obtain an optimal storage efficiency. The corresponding mapping rule has been implemented as a graph transformation rule shown in Figure 58, where

the graph nodes representing the subclass, superclass, and their inheritance relationship are mapped onto the node of the super class table.

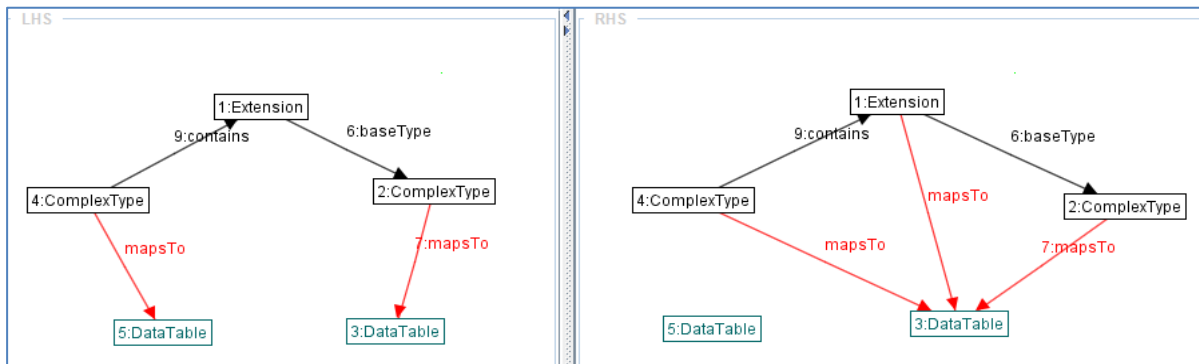


Figure 58: Rule 2: Mapping two ADE classes with inheritance relationship onto one table

According to the mapping concept explained in chapter 3.2.1, the prerequisites for performing this mapping rule has been formalized by three Negative Application Conditions (NACs) shown in Figure 59.

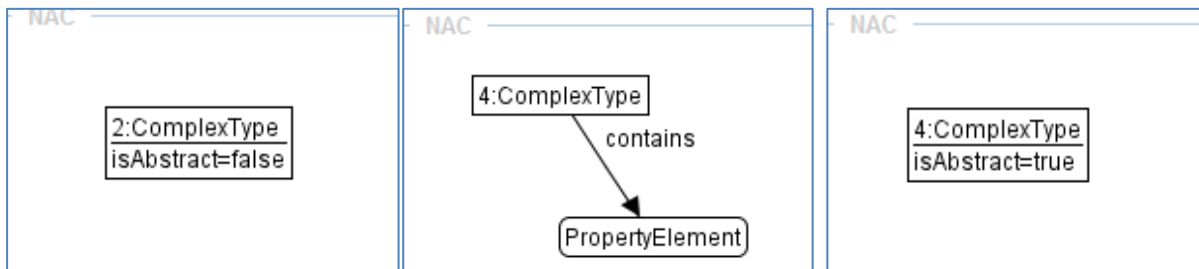


Figure 59: Negative application conditions of the Rule2

These three NACs specifies that, the mapping rule is only applicable, when the followings conditions are fulfilled:

1. The super class is an abstract class
2. The subclass does not have any attributes or associated with other classes
3. The subclass is not an abstract class

If the conditions are not satisfied, an additional mapping rule will be triggered, which maps the inheritance relation to a table join linking the two mapped tables using their primary key columns (cf. Figure 60).

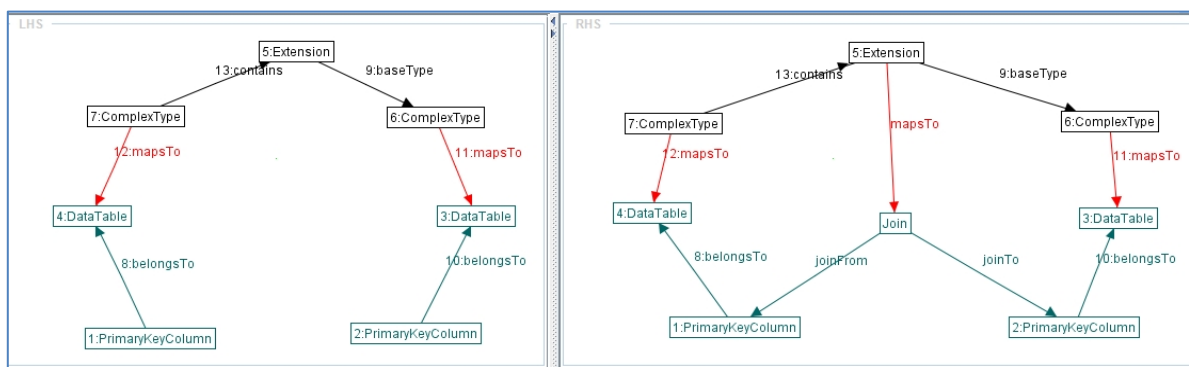


Figure 60: Rule 3: Mapping Inheritance to a foreign key constraint



In case that the superclass and the subclass are mapped onto one table and they also have a composition relationship (1:N association), a foreign key column for storing the parent object ID shall be added to the mapped table and referenced to its primary key column. According to the optimization approach introduced in chapter 3.2.2, the relational table structure can be further improved by introducing an additional foreign key column named `ROOT_ID` for holding the ID of the root element of each composition hierarchy. This allows the fast retrieval of all its child elements by just querying on the attribute `ROOT_ID` without the need to employ recursive database joins which can usually result in poor database performance. This mapping rule can also be expressed using graph transformation rule shown in Figure 61.

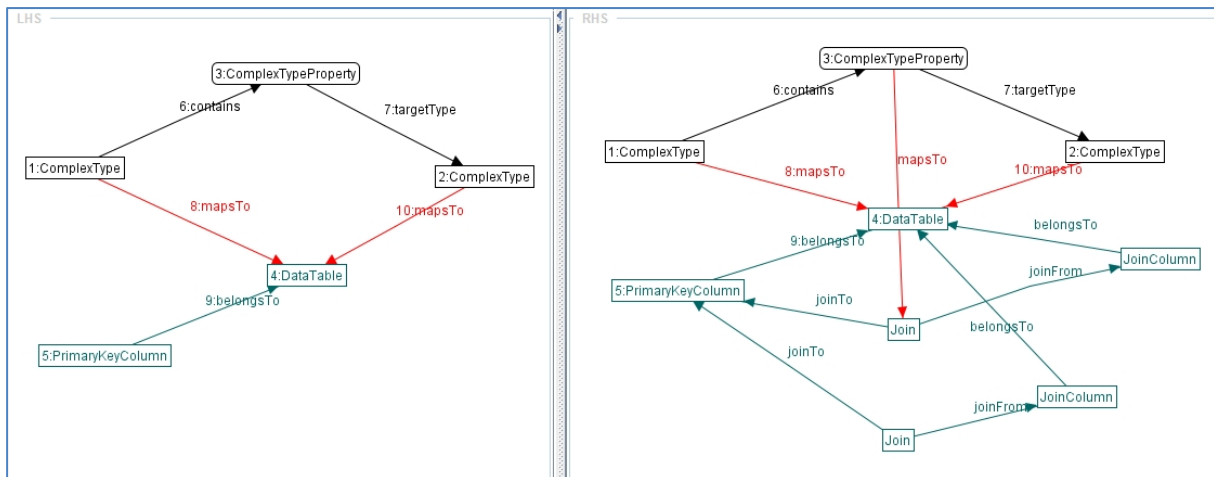


Figure 61: Rule 4: mapping two classes which have a composition relationship and are mapped onto the one table

Regarding the mapping of attribute properties of each class, an additional transformation rule (cf. Figure 62) has been designed for handling each simple data attribute which can be directly mapped onto a single table column with the corresponding primitive data type. The name of the mapped column could be equal to the attribute name.

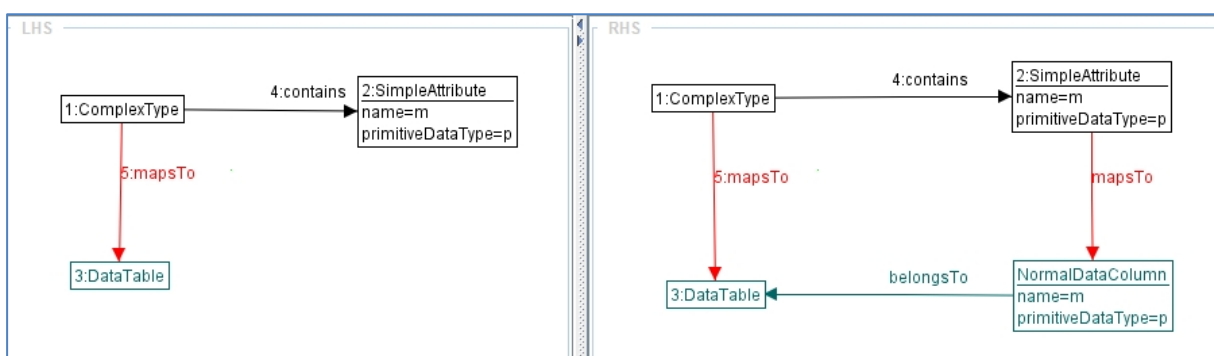


Figure 62: Rule 5: Mapping simple attribute property onto a table column

For handling the geometry property e.g. B-Rep-based surfaces and solids, the 3DCityDB table `SURFACE_GEOMETRY` shall be reused for storing the explicit geometry contents, because the compact structure (cf. chapter 3.2.2) of this table can guarantee the high database performance when accessing the complex-aggregated geometry objects from user applications. Another reason for collecting all B-Rep geometries in a single table is that CityGML and ADE feature types can have appearance information and it is much simpler to



only have to link the 3DCityDB APPEARANCE table to one single SURFACE\_GEOMETRY table.

The database implementation is realized using a foreign key column pointing to the primary key column of the SURFACE\_GEOMETRY table. To reflect this mapping concept in the graph transformation system, two graph transformation rules are need. The first one is shown in Figure 63, where a node representing the SURFACE\_GEOMETRY table shall be first created, if it does not exist already.

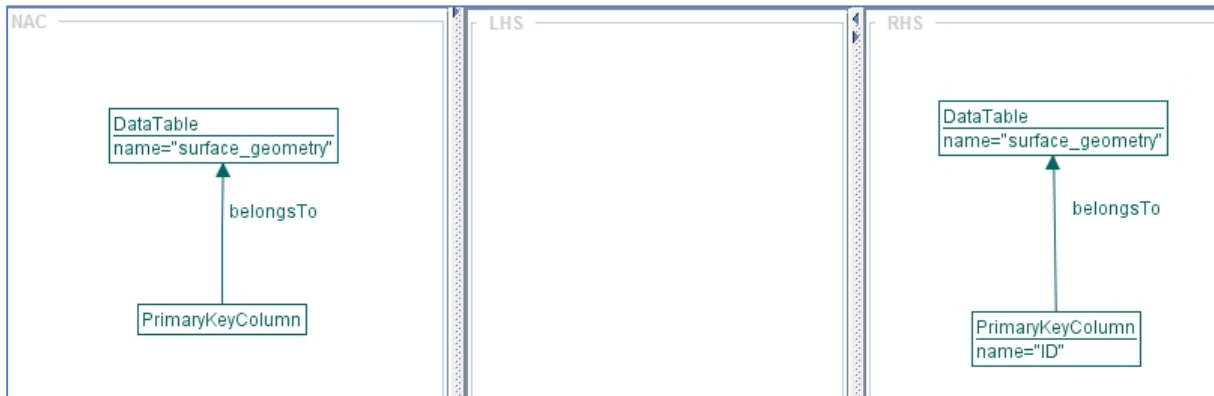


Figure 63: Rule 6: Initializing a node for representing the 3DCityDB table "SURFACE\_GEOMETRY"

Once the node of SURFACE\_GEOMETRY table has been initialized, a new node representing the foreign key column referencing to the SURFACE\_GEOMETRY table shall be created for the mapping of the B-Rep-based geometry property. The foreign key constraint holding the relations between the data table and the SURFACE\_GEOMETRY table is realized using a *Join* node. The corresponding graph transformation rule is shown in Figure 64.

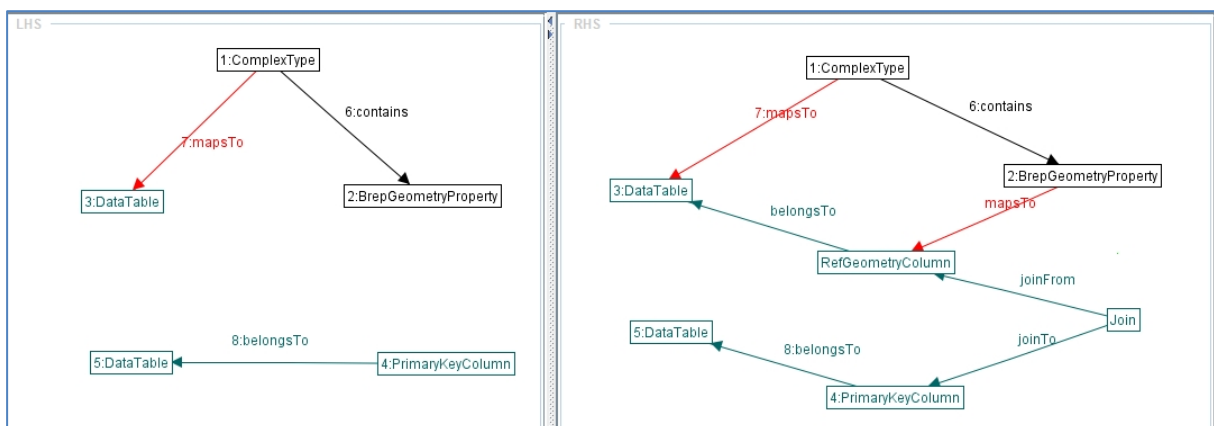


Figure 64: Rule 7: Mapping B-Rep-based geometry property onto a foreign key column referencing to the SURFACE\_GEOMETRY table

Above, only a subset of the developed graph transformation environment is presented, where a total number of 34 graph transformation rules have been designed for the automatic derivation of a compact database schema (see Appendix 1 for further graph transformation rules). These transformation rules are roughly grouped into the following layers which can be executed successively according to the numbered order.

1. **Layer 1:** Mapping each classes and complex data type onto a separate table along with a primary key column.

2. **Layer 2:** Mapping and merging classes with inheritance relationship to one table if certain conditions are fulfilled.
3. **Layer 3:** Mapping each inheritance relationship onto a foreign key constraint if the two classes with an inheritance relationship are not mapped onto the same table.
4. **Layer 4:** Mapping each association relationship (aggregation/composition) onto a foreign key constraint.
5. **Layer 5:** Mapping each attribute property to a column or to a foreign key constraint referencing to the table where the attribute contents are stored.
6. **Layer 6:** Creating a database index for each primary key and foreign key column and spatial data columns.
7. **Layer 7:** Creating a database sequence for each table mapped from a complex data type.

By applying such layered transformation rules, any ISO-19136-conform CityGML ADE model can be transformed to a compact relational database schema. An example application is given in the last subsection of this main chapter.

### 4.3.2 Extending the 3DCityDB for Managing CityGML ADEs

Once the relational database schema of an ADE is ready, the next task is to attach it to the standard 3DCityDB database schema for storing the CityGML instance documents containing the ADE data. In addition, the meta-information of the added ADE database schema should also be stored in the database in order to offer the possibility of facilitating the database administration e.g. the registration and deregistration of a selected ADE's XML schema (cf. Murthy et al. 2006). To reach this goal, the current 3DCityDB database schema (version 3.3.0) has been extended by decomposing all tables into three modules, namely *Metadata Module*, *Core Data Module*, and *Dynamic Data Module*, whose relations are shown in Figure 65.

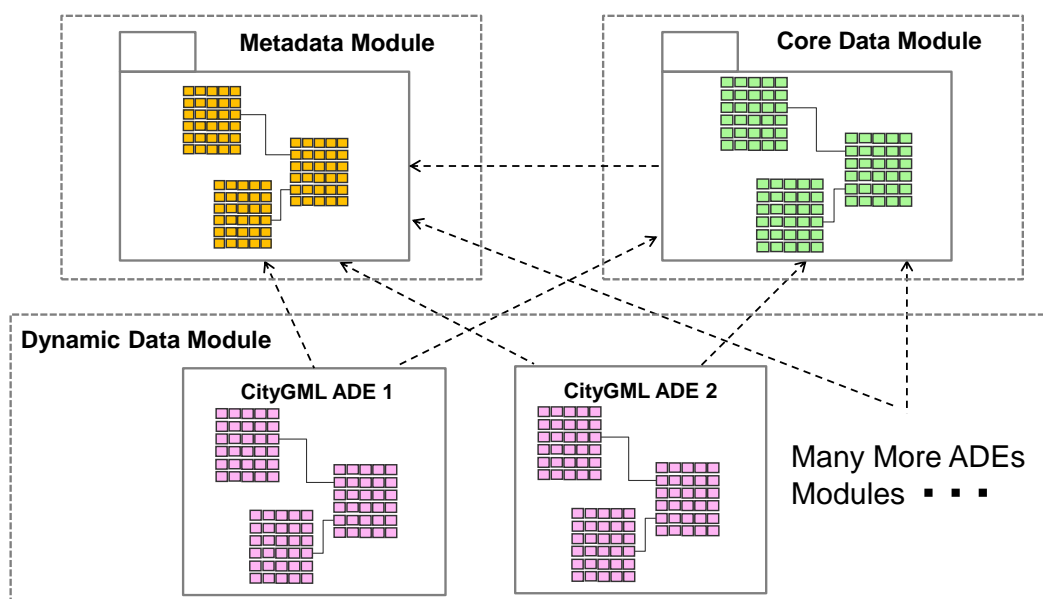


Figure 65: New conceptual 3DCityDB database structure for handling CityGML ADEs

The green grids enclosed in the *Core Data Module* represents those database tables that are already included in the current version of the 3DCityDB database schema which is responsible for storing the standard CityGML models such as *Building*, *Tunnel*, *Transportation*, *CityFurniture*, *CityObjectGroup*, *Generic*, *Appearance* etc. For a given CityGML ADE, an additional group of database tables forming a separate module belonging to the *Dynamic Data Module* (pink grids in Figure 65) shall be created and attached to the 3DCityDB database schema. In addition, the relationships (e.g. generalization/specialization and associations) among the model classes of CityGML and CityGML ADEs are adequately reflected using database foreign key constraints which can ensure the data integrity and consistency within the database system. The *Metadata Module* associated with the *Dynamic Data Module* is used for storing the relevant meta-information (e.g. the XML namespaces, schema files, and class affiliations etc.) about the application schema of the registered CityGML ADEs as well as the referencing relations among the ADE and CityGML schemas. This way, the dependencies between the registered ADE application schemas can be directly read from the 3DCityDB database schema itself which hence become to be dynamically manageable for handling multiple CityGML ADEs within a database instance. In the following, the implementation of the *Metadata Module* is presented in detail (cf. Figure 66).

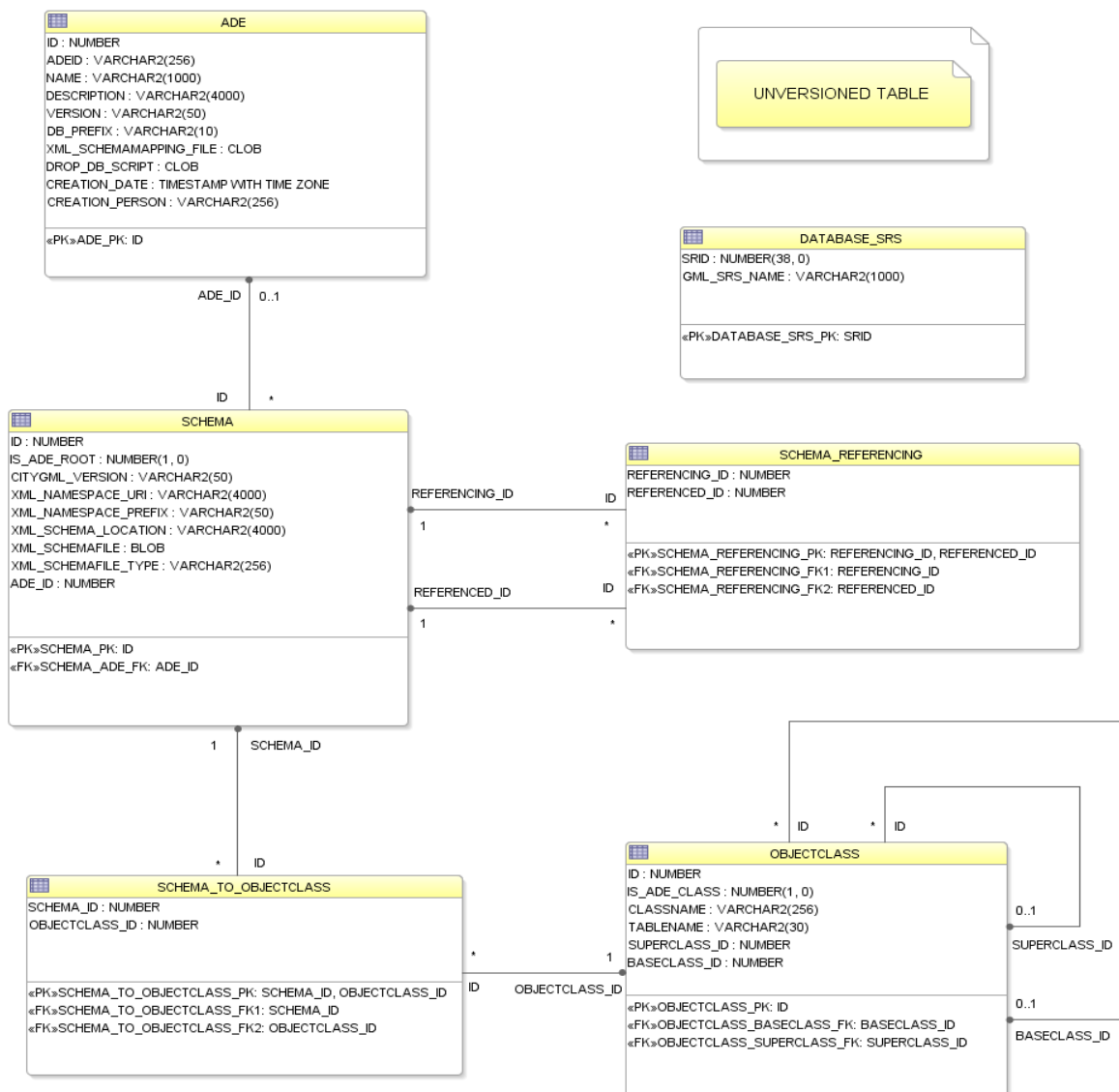


Figure 66: Technical implementation of the 3DCityDB Metadata Module in a relational diagram

The table ADE serves as a central registry for all the CityGML ADEs having been registered each of which is held in a table row and its relevant attributes are mapped onto the respective columns. For example, an ADE can be identified by assigning it with a globally unique ID which can be automatically generated as a UUID (Universally Unique Identifier). The columns NAME and DESCRIPTION are mainly used for storing the basic description information for each ADE. The column VERSION denotes the version number of an ADE and allows to distinguish different versions of the same ADE. In the 3DCityDB database schema, the database objects like tables, indexes, foreign key constrains, and sequences of a certain ADE shall be named by starting with a unique prefix text which allows applications to rapidly filter out the database schema of a chosen ADE using a SQL query with wildcard. In this way, it is possible to automatically perform some kinds of statistics on the ADE data contents that are stored in the individual tables. In addition, the column XML\_SCHEMAMAPPING\_FILE is used to store the XML-formatted schema mapping information of each ADE and has hence been defined with the CLOB data type. Another CLOB column is DROP\_DB\_SCRIPT where the SQL-script for dropping the ADE database schema is saved and can be easily retrieved to be carried out at the database side. Moreover, the CREATION\_DATE and CREATION\_PERSON are two application-specific attributes for providing the information about when and who has registered the individual ADE within the database to facilitate the database administration work, e.g. deletion of the outdated ADEs from the database.

A CityGML ADE may consist of multiple application schemas one of which should be the root schema that references the others. Such dependency information along with the meta-information of the individual schema are stored in two tables, namely SCHEMA and SCHEMA\_REFERENCING. The SCHEMA\_REFERENCING table is an associative table which contains two foreign key columns REFERENCED\_ID and REFERENCING\_ID to link the respective referencing schema and referenced schema. In the table SCHEMA, the flag attribute IS\_ADE\_ROOT is used for denoting the root schema that directly or indirectly references all the other ADE schemas of an ADE. In this way, the dependency hierarchy of the ADE schemas can be fully represented in a relational model to facilitate the reconstruction of the original schema relations through user applications. For each schema, its relevant meta-information such as the schema location, namespace, namespace prefix, source schema file, as well as the file type of the schema can also be stored in the remaining columns of the SCHEMA table. Additionally, since an ADE schema can be developed based on either the CityGML version 1.0.0 or 2.0.0, the column CITYGML\_VERSION has been introduced to denote this variable.

The table OBJECTCLASS is a central registry for enumerating the standard CityGML classes along with the classes of the registered ADEs. Each class shall be assigned with a globally unique numeric ID for querying and accessing the class-related information. According to the 3DCityDB implementation, the integer values ranging from between 0 and 106 have already been reserved for the standard CityGML classes. Thus, the ID values of the registered ADE classes must be larger than 106. However, concerning the future versions of the CityGML standard into which more additional feature classes might be added, a certain range of integer values must be preserved and shall also not be used for ADEs. Therefore, for each ADE, it is recommended to assign its classes with a set of large and incremental integer values which can be started with 10000. In order to avoid the class ID conflict, each ADE shall own a

certain large value range which can be centrally maintained by an official community like the 3DCityDB organization group. The OBJECTCLASS table also contains a few additional columns like the IS\_ADE\_CLASS which is a flag attribute to denote which classes come from ADEs. Another column named TABLENAME refers to the table name of a CityGML or ADE class to provide a simple model mapping information. The last two columns SUPERCLASS\_ID and BASECLASS\_ID are two foreign key columns of the ID column. The SUPERCLASS\_ID points to the parent class, and the BASECLASS\_ID reflects the base class which can be *ComplexType*, *AbstractGMLType*, or *AbstractFeatureType* (cf. chapter 2.4). These two columns allow representing the inheritance hierarchy of all the CityGML and ADE classes in a relational structure.

To perform the ADE registration, the CityGML Import/Export tool has been extended with a newly developed plug-in called ‘ADE-Manager’ (cf. Figure 67) which provides a graphical interface allowing users to register a selected ADE into the database and at the same time to execute the automatic creation of the database scheme having been derived using the graph-based converter tool (cf. chapter 4.3.1). To establish the database connection, the core module of the Import/Export tool can be used which exposes a number of APIs to the plugins for invoking the functions like database control, multi-threading support, event dispatching, and plugin management etc. Once the database connection has been done, the schema mapping file shall be read which is able to contain the relevant meta-information for the ADE registration into the 3DCityDB’s *Meta Data Module*. In the subsequent step, the SQL script of the ADE database schema shall also be read in order to automatically perform the creation of the database objects like tables, indexes, and sequences, and foreign key constraints. To check the results of the ADE registration, it is also possible to list the meta-information of all the registered ADEs on the graphical user interface by querying the contents from the ADE table directly.

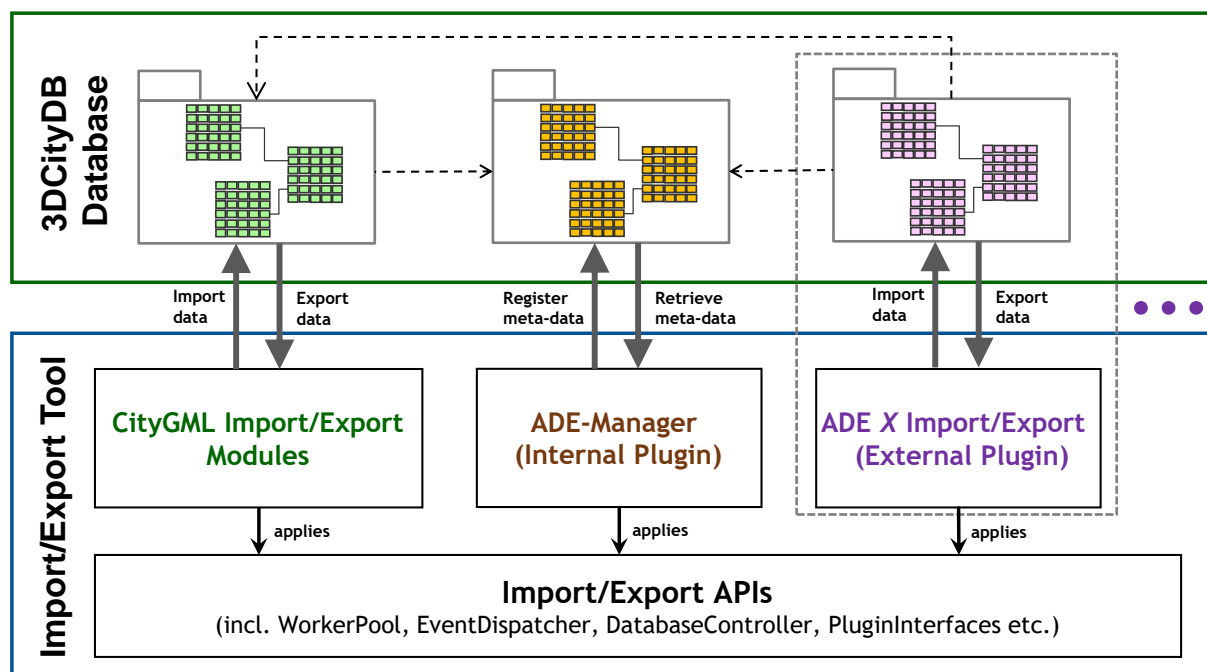


Figure 67: Software structure of the extended CityGML Import/Export Tool

In order to realize the import and export of CityGML ADE datasets, additional plugins for the Import/Export tool have to be developed for the individual ADEs. These plugins do not

require graphical user interfaces, but the functionalities of reading the ADE contents contained in the CityGML documents to write them into the database. Such functionalities can be implemented by extending the citygml4j library which will become capable of handling not only the CityGML data elements but also the ADE's. For example, while importing a given CityGML document, the CityGML features will be processed as usual using the standard CityGML Import module. Once an ADE feature is visited, it will be passed to the ADE plugin whose import functions will be automatically invoked for performing the ADE data import.

Since an Import/Export tool can be connected with multiple database instances, it might occur that the installed ADE Import/Export plugins are not consist with the ADEs registered in the target database. For example, an ADE plugin has been already installed at the client side but the respective ADE has not been registered into the connected database. In this case, the Import/Export tool must perform a match-check when it is launched. This match procedure can be done by following two steps. First, the Import/Export tool should determine which ADE plugins have already been installed and check whether they are runnable by validating them against the plugin interface. If invalid ADE plugins have been found, a runtime exception shall be raised through an alert window to show the detailed error message. If all installed ADE plugins have passed this validation procedure, the Import/Export Tool will display an overview list of the supported ADE plugins. After connecting with the target database, the second checking step should be immediately started by the Import/Export tool to determine which ADEs have already been registered into the database. To realize this, a list of the registered ADEs shall be fetched by querying the values of the ADEID columns in the ADE table. Since the ADE ID can be seen as a fingerprint of a physical ADE, it can be utilized to carry out a short test to match each installed ADE-plugin to the registered ADEs. The names and descriptions of all unmatched ADEs should be displayed on the application window to guide users in installing the missing ADE plugins. With the help of the two-steps check process, the Import/Export tool is able to guarantee the successful import and export of the ADE datasets.

The last major aspect in the context of the CityGML ADE management is deregistration and deletion of the data contents from the database. This requires three processing steps which are shown in the Figure 68. First, for a given ADE, a list of the corresponding ADE class IDs can be easily retrieved from the OBJECTCLASS table where all CityGML and ADE classes are registered. It is hence possible to iterate through the fetched ID list and perform a simple SQL deletion command like

“DELETE from CITYOBJECT where OBJECTCLASS\_ID = [ID]”

Where the variable [ID] stands for each ADE class ID. By means of the database foreign key constraints, the CASCADE-deletion mechanism can be used to automatically delete all ADE data contents except those that are stored in the CityGML tables which are linked with the ADE data tables via associative tables, because the CACADE-deletion actions can only be applied to the referenced table being directly linked with the referencing table via a foreign key constraint. In order to overcome this issue, the database trigger mechanism can be used to execute the complete deletion in response to the deletion event on the referencing table. In the second step, the ADE database schema including the tables, indexes, sequences, and foreign key constraints shall be dropped from the database by running the DROP\_DB script which

can be retrieved from the column `DROP_DB_SCRIPT` in the ADE table. In the last step, all meta-information of the target ADE shall be removed from the metadata tables to complete the ADE deregistration. This can be done by simply deleting the corresponding record from the ADE table, because the `CASCADE`-deletion can be applied to all the metadata tables according their relational structure (cf. Figure 66).

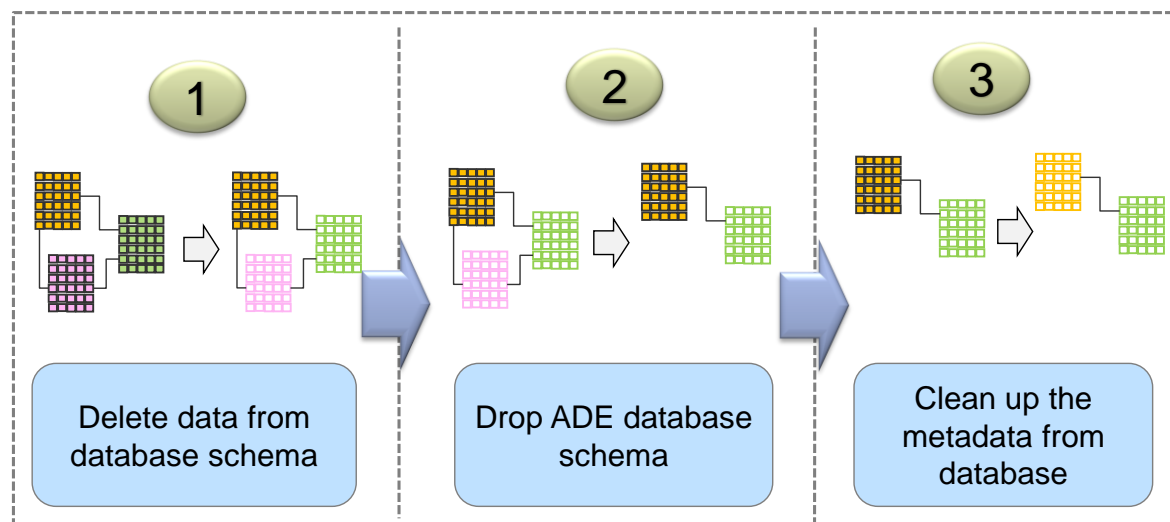


Figure 68: Workflow of deregistration for a CityGML ADE from a database instance

### 4.3.3 Example Application: CityGML-TestADE

For testing and evaluation of the research results and the implemented software tools in the context of the thesis, a specific CityGML ADE called ‘TestADE’ has been carefully developed from scratch to provide a comprehensive testbed for simulating different scenarios which can frequently occur in practical applications. It can hence be seen as an artificial ADE which is not aimed to be employed for a specific domain application, but can be flexibly modified in response to the various test needs. In order to experiment the optimization approaches presented in the section 3.2.2 using the developed converter tool, this test ADE includes a couple of feature classes and complex data types which are originated from different existing CityGML ADEs such as Energy ADE and UtilityNetwork ADE. Note that since the optimizations are mainly carried out by evaluating the relations between classes, most attributes and properties of the original classes are omitted and only a few ones are kept. This can make this artificial ADE very light-weight and intuitive, which allows to facilitate the software tests as well as the graphical representation of the models. For the sake of clarity, all classes of this ADE have been categorized into the following groups which are painted with different colors in a UML diagram (cf. Figure 69).

- The classes painted with blue color represent the standard CityGML classes such as *AbstractCityObject*, *AbstractSite*, *AbstractBuilding*, *BuildingPart*, *Address*, *AbstractBoundarySurface*, and *BuildingRoofSurface* which are related with the defined ADE classes.
- The class *AbstractBuilding* with the stereotype `<<ADEElement>>` is a specific class for holding all ADE hook attributes of its super class, namely the standard CityGML class *AbstractBuilding*. The two classes are related using an inheritance relationship



which is tagged with a stereotype <<ADE>> to indicate that this inheritance relationship is not equivalent to the normal inheritance in the context of object-oriented modelling.

- The classes, which are directly or indirectly derived from the standard CityGML classes, can be recognized by their yellow color. This class group includes the classes like *IndustrialBuilding*, *IndustrialBuildingPart*, *IndustrialBuildingRoofSurface*, *AbstractBuildingUnit*, *BuildingUnitPart*, *BuildingUnit*, and *OtherConstruction*.
- The pink classes like *DHWFacilities*, *LightingFacilities*, and *Facilities* are those classes which are not derived from the existing CityGML classes but from the GML feature class.
- The green classes represent the geometry types, which have been defined according to the ISO 19107 standard. The three typical geometry types e.g. *GML\_Solid*, *GM\_MultiSurface*, and *GM\_MultiCurve* have been chosen for representing the geometric information.
- The last category is the class *EnergyPerformanceCertification* which is not a feature class, but a complex data type defined with the GML stereotype <Datatype> for representing a collection of simple attributes.

Based on this ADE model, a compact relational database model can be obtained according to the mapping rules introduced in the section 4.3.1. The concrete mapping idea is graphically shown in Figure 70, where the classes are grouped with a set of blocks each of which stands for a merged table. In addition, both the Oracle and PostGIS compliant database schemas along with an XML-based schema mapping file have been successfully generated whose SQL and XML contents are printed out in the appendixes of the thesis. Further, a 3DCityDB database instance was created on top of a PostGIS database instance on which the corresponding SQL scripts have been executed to create the respective ADE database schema. By means of the open-source software tool pgModeler, a reverse engineering process has been carried out to generate a relational diagram (cf. Figure 71) of the ADE database schema. A total number of 16 tables are displayed in the database diagram, where the tables painted with green color represent those 3DCityDB tables that already exist for the mapping of the standard CityGML classes. The yellow table OBJECTCLASS is the metadata table which is also a default table shipped together with the standard package of the 3DCityDB. The other 10 tables painted with green color correspond to the dynamically extended ADE database schema which has been automatically derived according to the optimization concept sketched in Figure 70. For highlighting the advantage of the achieved result, a comparison to the database schemas produced from different modern tools like ShapeChange, and Deegree has been made by evaluating the number of the created tables. The comparison result is listed in Table 2.

Table 2: Quantitative comparison of the number of tables generated from different software tools

	Graph-based Converter Tools	ShapeChange (version 2.3.0)	Deegree (version 3.3.20)
Number of Tables	10	21	60



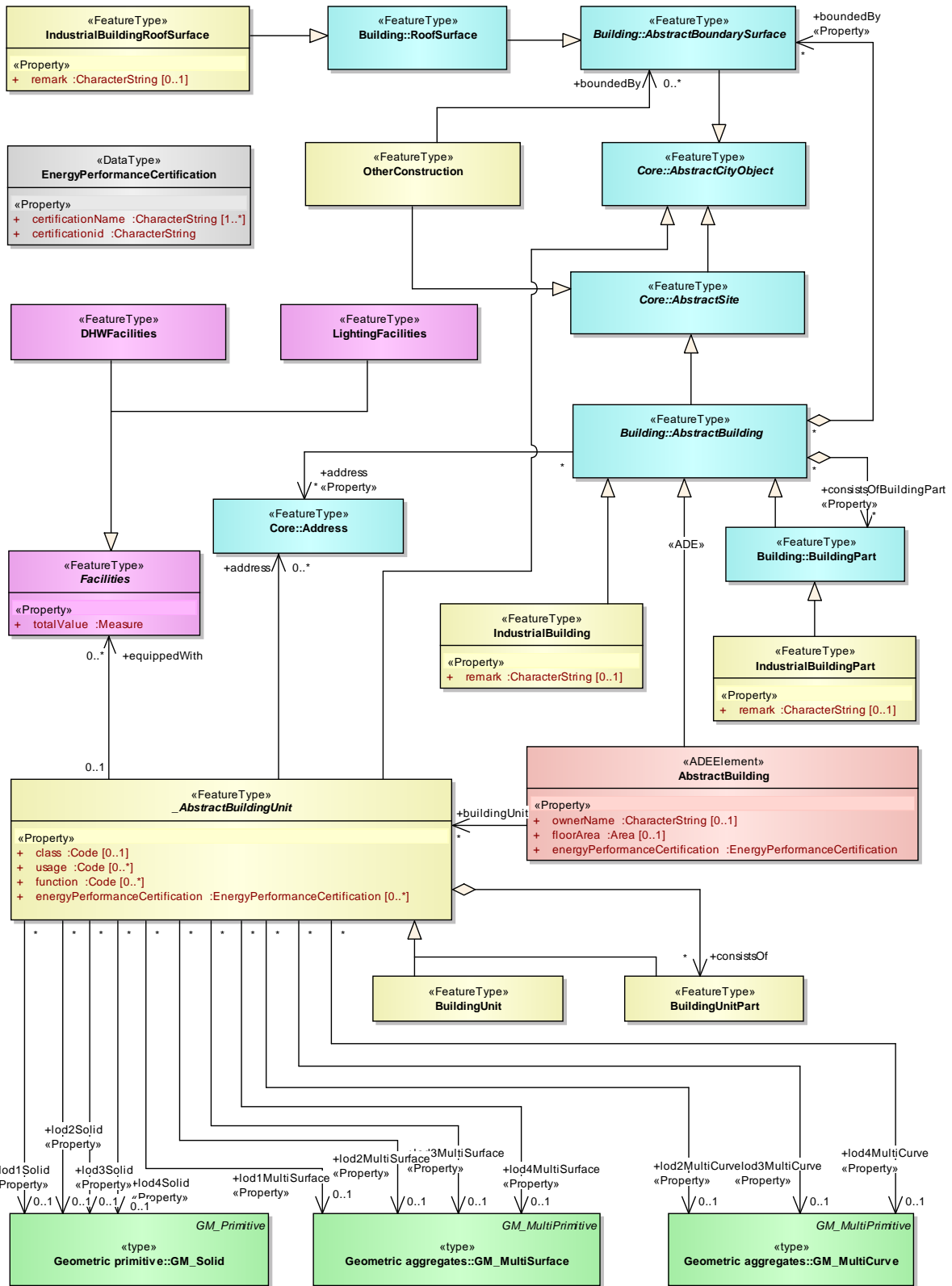


Figure 69: An artificial CityGML ADE for testing the developed graph-based transformation approach

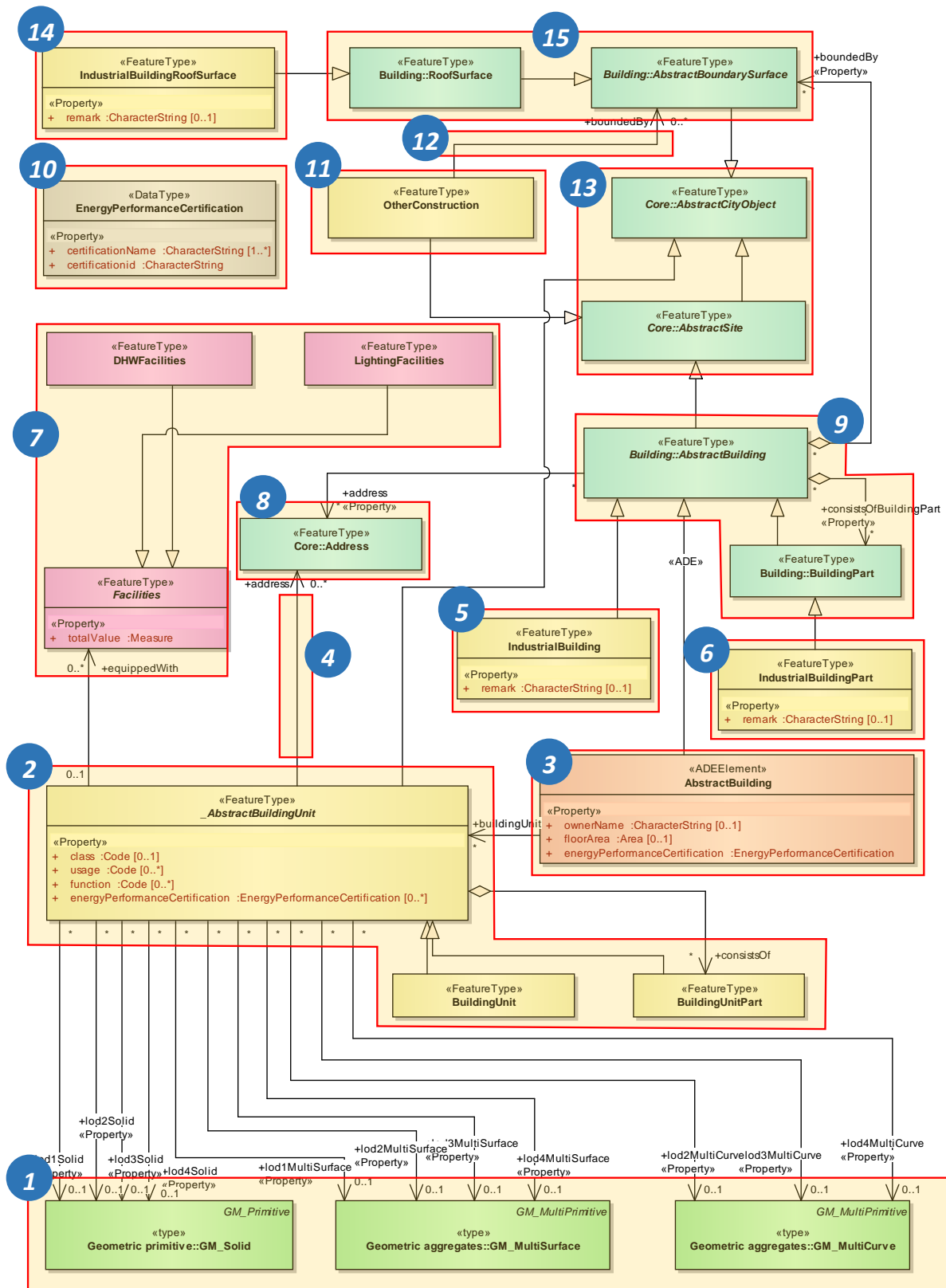


Figure 70: Concept of grouping and merging GML, CityGML, and ADE classes for deriving compact relational database models. Note that the classes of the groups 1, 8, 9, 13, and 15 are standard GML and CityGML classes, which have already been mapped to the default 3DCityDB tables.

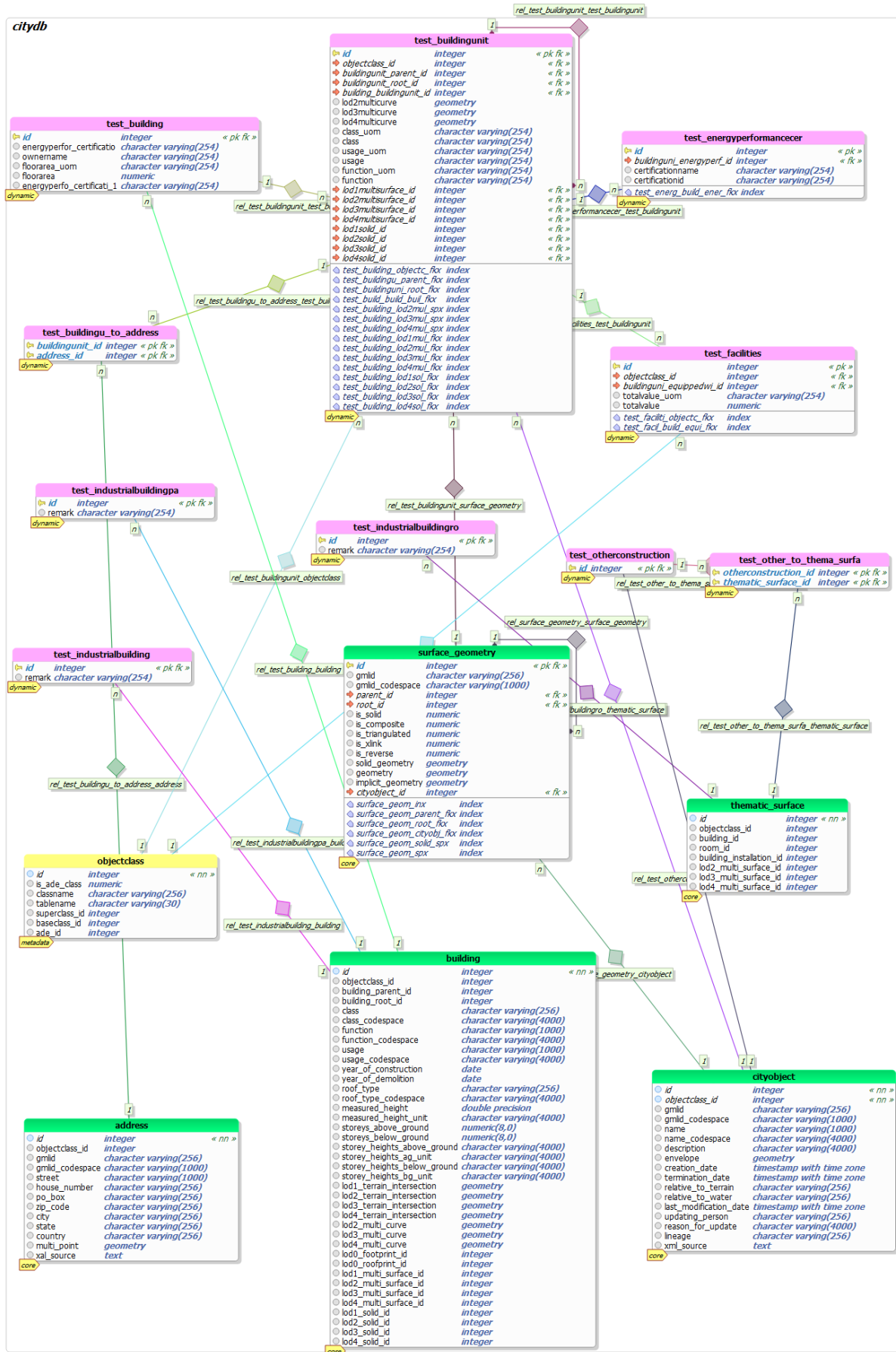


Figure 71: Relational diagram of the automatically derived relational database of the artificial ADE using the developed graph-based converter tool

#### 4.3.4 Practical Applications: EnergyADE and UtilityNetworkADE

The developed graph-based converter tool has also been successfully tested with the derivation of 3DCityDB-compliant relational database schemas for existing practical CityGML ADEs e.g. EnergyADE and UtilityNetworkADE. The EnergyADE is an open and standardized CityGML ADE that defines a comprehensive data model for facilitating the energy analysis and simulations in building and urban scale (cf. Agugiaro et al. 2018), while the UtilityNetworkADE extends CityGML to allow for modelling different types of networks e.g. electricity, wastewater, gas, and telecommunication networks in detail with respect to their topographical, topological, and functional aspects (cf. Kutzner & Kolbe 2016). Both ADEs are currently being actively developed at the time of writing the thesis and the developer groups of both ADEs are converged to collaboratively work together with the common aim of optimizing the data models.

Both ADEs have more complex model structures regarding the technical implementation of their relational database schema. For instance, the current version of the UtilityNetworkADE contains a total number of 53 feature classes and complex data types some of which are constructed with complex class relationships e.g. Composite design pattern and include thematic and spatial properties. Using the graph-based converter tool, the UtilityNetworkADE model can be mapped onto a relational database model composed of 36 tables. While testing the EnergyADE, 72 tables have been derived from 74 classes and complex data types. Both derived ADE database schemas have been successfully registered into one 3DCityDB instance which is able to provide the basis for realizing the interoperable analysis and simulation across the both application domains. Figure 72 shows an excerpt of the summary of all tables of the standard 3DCityDB and registered ADE database schemas. This summary report can be automatically generated using the CityGML Import/Export Tool after connecting to the database server.

The screenshot shows the 'Database' tab of the CityGML Import/Export Tool. The 'Connection' section is configured for 'TestADE' with a PostgreSQL/PostGIS type, localhost server, and port 5432. The 'Database operations' section includes a 'Generate database report' button. The 'Console' window on the right displays a list of tables, categorized into three groups:

- EnergyADE tables (excerpt):** #NG\_SOLARTHERMALSYSTEM, #NG\_SOLIDMATERIAL, #NG\_STORAGESYSTEM, #NG\_SYSTEMOPERATION, #NG\_THER\_DELI\_TO\_THER\_BOUN, #NG\_THERMA\_TO\_THERMA\_SURFA, #NG\_THERMALBOUNDARY, #NG\_THERMALDISTRIBUTIONSYS, #NG\_THERMALOPENING, #NG\_THERMALSTORAGESYSTEM, #NG\_THERMALZONE, #NG\_THERMALZONE\_TO\_ROOM, #NG\_TIMESERIES, #NG\_TIMEVALUESPROPERTIES, #NG\_TRANSMITTANCE, #NG\_USAGEZONE, #NG\_VOLUMEYPE, #NG\_WEATHERDATA.
- CityGML tables (excerpt):** #OPENING, #OPENING\_TO\_THEM\_SURFACE, #PLANT\_COVER, #PASTER\_RELIEF, #RELIEF\_COMPONENT, #RELIEF\_FEAT\_TO\_REL\_COMP, #RELIEF\_FEATURE, #ROOM, #SOLITARY\_VEGETAT\_OBJECT, #SURFACE\_DATA, #SURFACE\_GEOMETRY, #TEX\_IMAGE, #TEXTUREPARAM, #THEMATIC\_SURFACE, #TIN\_RELIEF, #TRAFFIC\_AREA, #TRANSPORTATION\_COMPLEX, #TUNNEL, #TUNNEL\_FURNITURE, #TUNNEL\_HOLLOW\_SPACE, #TUNNEL\_INSTALLATION, #TUNNEL\_OPEN\_TO\_IHEM\_SRF, #TUNNEL\_OPENING, #TUNNEL\_THEMATIC\_SURFACE.
- UtilityNetworkADE tables (excerpt):** #UT\_BUILDING, #UT\_CABLE, #UT\_CANAL, #UT\_CHEMICALCLASSIFIED.

Figure 72: Report of Database instance including EnergyADE and UtilityNetworkADE database schemas

## Chapter 5 Visualization and Exploration of Large Semantic 3D City Models

The 3D visualization and exploration of CityGML-based semantic 3D city models along with domain-specific extensions is very essential to facilitate inspecting, manipulating and analyzing 3D geo-information. On the one hand, the 3D visualization application builds an interface between the users and the databases where the CityGML data are stored for realizing easy data access. The 3D visual representation of the real-world objects allows broad users to explore and validate the spatial contents of the urban information in an intuitive way without needing to inspect the original XML data contents in the database. On the other hand, the data management i.e. data modification, deletion, and enrichment can also be facilitated by the visual interaction with the 3D city model objects owning complex geometric or topological structures. To develop a sophisticated application supporting efficient 3D visualization and exploration, a number of challenging issues are encountered. First, CityGML documents usually carry the data on urban scale and are hence very large in size which can result in a significant performance issues on those hardware that have limited computing resources and capabilities. In addition, since the CityGML data model may have very complex data structures due to the increasing number of the attached domain extension information models, the exploration of, querying on and interaction with such complex 3D city objects are difficult to be operated by the application users who are not familiar with the CityGML standard.

Nowadays, none of the above-mentioned issues have been fully answered or solved by the existing software solutions. For example, a number of desktop-based 3D viewer applications i.e. Autodesk LandXplorer, FME Data Inspector, FZKViewer, and Aristoteles etc. are available which support the 3D visualization and exploration of CityGML data. However, these software applications can only be operated on certain operating systems and have no support for mobile devices. They can only be operated on certain machines where they are installed and hence must be installed on every computer for the application users. This will hinder the broad use of the software applications, since the installation of these software tools mostly requires administrative privileges and cannot be done by the normal users. In addition, all these software tools are limited in the capability of handling large CityGML documents and are only able to render CityGML datasets with a file size not exceeding the available computer memory. Moreover, in most GIS applications, CityGML data are provided by different data providers and are distributed from different servers which requires the functionality of accessing the data over the Internet which is also not supported by these software applications.

This limitation can be overcome by realizing the 3D visualization through web-based applications, which just need to be deployed on a web server at one time and can be accessed and operated on web browsers and platforms over the Internet (cf. Altmaier & Kolbe 2003). With the growing capabilities of modern web browsers due to the technological advancement of HTML5 and WebGL, more and more browser-based 3D applications have been developed which not only show a high performance but also can be run on all major operating systems which just require the installation of a web browser without needing to pre-install any software applications or browser plugins. Nowadays, since almost all mobile devices like tablet PCs and smart phones with modern mobile operating systems support HTML5 and

WebGL, the cross-platform 3D visualization and exploration of semantic 3D city models became possible (cf. Figure 73).

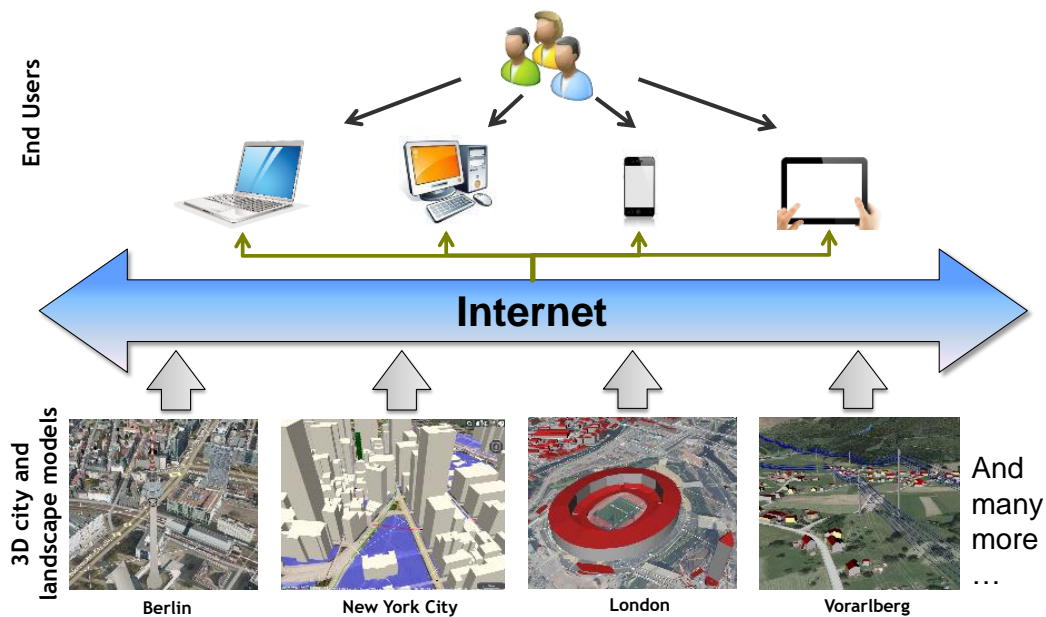


Figure 73: Conceptual structure for the Web-based geo-visualization of 3D city models

The main objective of this chapter is to answer the research and development challenges on how to utilize the web-based approach for realizing the high-performance 3D visualization and interactive exploration of semantic 3D city models according to the CityGML standard. In this context, a developed web-based solution for pre-processing, visualization and analysis of very large semantic 3D city models is introduced. In the first subsection, main focus is placed on the approach for the efficient 3D visualization of large 3D city models according to the CityGML standard. The second subsection introduces a 3D web client allowing not only for 3D visualization but also for the rich exploration and data analysis by linking with thematic information by means of Cloud-based technologies. In the last subsection, three typical application examples are shown to demonstrate the usability of the developed approach.

## 5.1 Visualization of Semantic 3D City Models

CityGML was primarily designed as an integrative data model for carrying a wide range of urban information and its instance documents are usually created and provided as a single document with a large file size in an XML structure. This data structure is optimal for the purpose of data transmission and exchange but not sufficient for 3D visualization in most GIS mapping application. For example, in case of CityGML building objects, their outer shell and shape are represented using explicit solid or boundary geometries along with the XLink mechanism. Such model representation is sufficient to reflect the geometric and topological information but not optimal for the 3D computer visualization compared to 3D graphics models which typically use scene graph concepts like grouping concept or triangulation of the model vertices. Second, CityGML is actually not a visualization format and was not designed for the efficient representation of the object appearances such as graphical materials and textures in viewing applications. Although this visual information can be semantically modeled inside a CityGML dataset using its Appearance module e.g. a huge texture atlas,



extra processing steps are required for splitting this texture image and drape each split fragment onto the corresponding object surface and may consume more computation time. Moreover, the rendering of a very large CityGML dataset at a time can also easily cause memory overload problem. Thus, the CityGML datasets shall be converted to an efficient 3D graphics format with a tiled data structure in order to realize the high-performance 3D data visualization on the different platforms to archive a better viewing experience.

### 5.1.1 Creation of 3D Visualization Models

For the efficient 3D visualization, a pre-processing step is required for the conversion of CityGML 3D models to 3D visualization models using 3D graphic formats. Such data formats can be efficiently read and parsed by web browsers and computer graphic cards for fast rendering of the 3D contents in the web browsers. The processing pipeline is summarized in Figure 74.

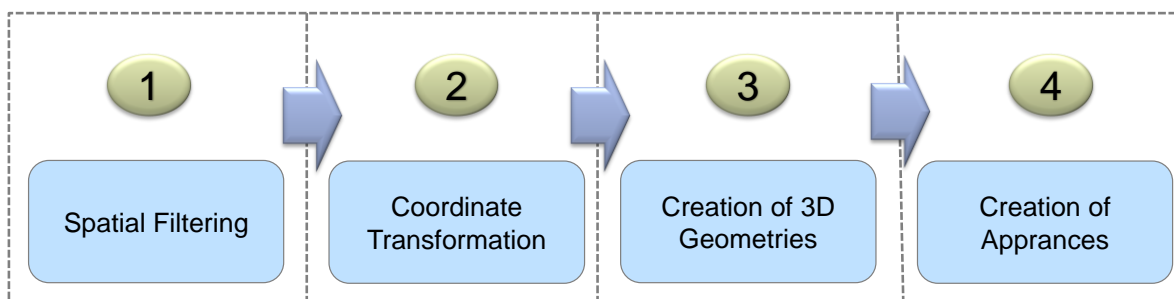


Figure 74: Workflow of generating 3D visualization models

Since a CityGML dataset usually contains the information with the full coverage of a city and only parts of the data are needed for specific application cases, a subset of the data shall be first extracted. This can be done by applying a combination of different filter criteria such as spatial bounding box, thematic feature classes, or the levels of detail. For example, by using the thematic filter, different feature classes such as buildings, trees, water, and transportation objects etc. can be generated separately to create a set of different thematic data layers each of which can be activated or deactivated on the client application individually. With the spatial filter, all objects of the selected feature classes, which are lying within or intersected with a chosen spatial bounding box, can be exported when, for instance, exporting a certain district from the entire city model. Moreover, since CityGML allows to represent the same object with different levels of detail at the same time, multiple 3D visualization data layers align with the available detail levels can be generated. In this way, the 3D viewer applications can dynamically load the same 3D object at different details according to the distance to the current camera position to optimize the rendering performance.

The second transformation step is the coordinate transformation. Although the geometric information of CityGML datasets can be represented using different types of coordinate reference systems (CRS) such as 3D geographic CRS, 3D Cartesian CRS and 3D projected CRS etc., the spatial elements in CityGML models are mostly represented using a projected CRS such as such UTM and Mercator etc. or a compound CRS which combines a projected CRS with a vertical CRS for the height reference (cf. Figure 75). Since most web mapping engine like Cesium and Google Earth virtual globes only support the geographic coordinate

system according to the WGS84 ellipsoid, a coordinate transformation process is hence additionally required. This transformation process can be directly performed using the spatially-enhanced database management systems which usually provide full support for converting different coordinate reference systems.

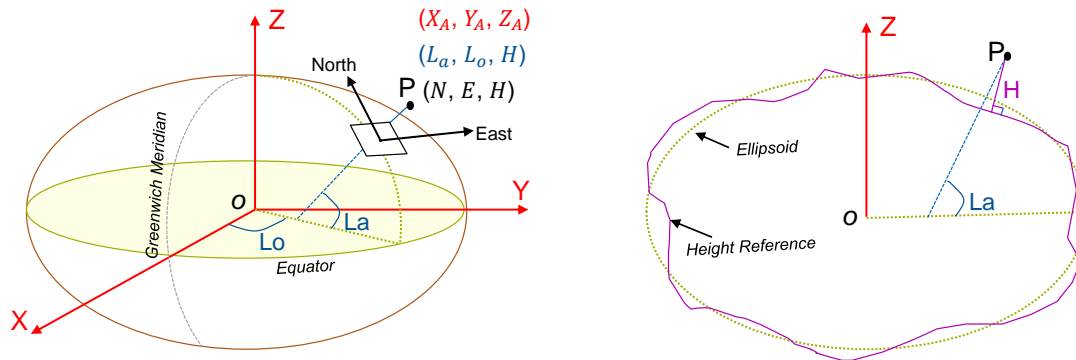


Figure 75: Comparison between the coordinates of different reference system types. Note that the latitude ' $L_\alpha$ ' of the point ' $P$ ' is geodetic latitude which is the angle between the corresponding surface normal and the equatorial plane, rather than the angle (called 'geocentric latitude') between the equatorial plane and the line connecting the point ' $P$ ' and the origin ' $O$ '.

After performing the coordinate transformation, the horizontal coordinates of the 3D model objects can be correctly transformed to the target position as geographic latitude and longitude. The original height, which is usually based on the vertical reference surface (cf. Figure 75), may remain unchanged according to the vertical height reference system. When viewing the 3D object on a 3D virtual globe, a digital terrain model (DTM) is needed to ensure a proper display of the 3D models on the Earth surface. However, the 3D models may hover or sink into the ground in some unusual cases, i.e. wrong height data of the source data or low resolution of the DTM being loaded into the 3D virtual globe. To overcome this issue, one of the simplest solutions is to choose the lowest point of the 3D object as the anchor or reference point and adjust it to the height value 0. The 3D virtual globe must interpret the height value as relative height above the loaded DTM surface to automatically render the 3D object on the terrain surface according to the relative height. However, virtual globes sometimes do not provide such "relative altitude mode" functionalities and are hence not able to automatically perform the height adjustment on-the-fly.

The second solution is to move the bottom of each object onto a certain DTM surface by adjusting the height value of its anchor point by calculating a so-called z-Offset value (cf. Kolbe et al. 2016). This value can be obtained from subtracting the anchor point's z-coordinate from the DTM's elevation value of the same planar position. To realize this, the elevation value of the DTM for each given position must be accessible. For example, Google provides a so-called Google Maps Elevation API which provides a world-wide elevation data and allows users to fetch the height value of a given position through the HTTP interface by constructing a simple GET-request (cf. Google Elevation API 2017). However, the provided DTM has relatively low resolution for most areas and this API also imposes strong usage restrictions for non-premium users who can only issue a maximum of 2500 requests per day. In practical applications, this limit can be easily reached when generating massive 3D models for large areas. Therefore, the users must deploy their own elevation API by storing the DTM data into a spatially-enhanced database which should be able to perform the coordinate



transformation to convert the world coordinates of the selected position to the corresponding raster space for querying the respective elevation value from the target DTM. However, with the increasing size of the DTM data, querying of the elevation value for every 3D object's anchor point may slow down the overall data generation process.

While performing the export process, the calculated `zOffset` can be stored as a CityGML generic attribute in the `CITYOBJECT_GENERICATTRIB` table of the 3DCityDB instance. In this way, subsequent exports will be faster since the `zOffset` value is already stored in the 3DCityDB and does not have to be calculated again. Since city objects may have different geometries for different LoDs, the anchoring points and their elevation values may also differ for each LoD. Thus, for each LoD of a single object, an individual `zOffset` attribute is required. Saving the building's height offset in the form of a generic attribute ensures that this information will be stored in exported CityGML datasets and can thus be transmitted across different database instances (cf. Figure 76).

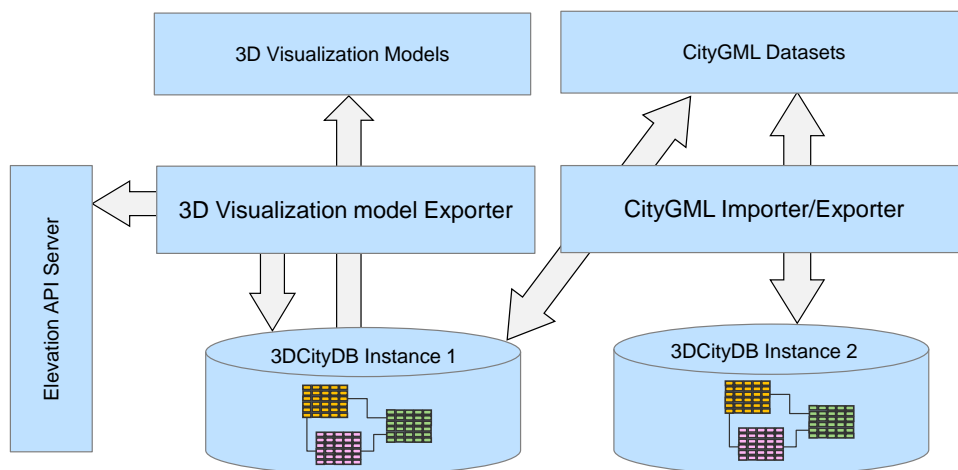


Figure 76: Workflow of interacting with Elevation API

According to the CityGML specification, the same city object can be represented using different geometries on different Levels of Detail ranging from 0 to 4. These geometry information can directly be used for the generation of the 3D visualization models for different LoDs. However, in practical applications, the city models usually do not contain the geometry information for all LODs, but just contain the geometries of the highest available LOD. The reason is that city object on higher LODs can be easily generalized to the lower ones by means of an automatic process (cf. Mao et al. 2011). This makes the data storage and management of 3D city models much more efficient. However, when generating a 3D visualization model, a top-down geometric simplification process is required to generate different display forms with different geometry details which are needed for certain application scenarios. There are four typical display forms (cf. Figure 77) which are often used in practical applications. These display forms are *Footprint*, *Extruded*, *Geometry*, and *Geometry with Textures* that grow in the degrees of geometric information details (cf. Kolbe et al. 2016):

- **Footprint:** 3D objects can be simplified to 2D representations by projecting the whole 3D bodies onto planar 2D polygons. Such simplified geometry representation can massively reduce the geometric complexity of the original 3D models and just display

their 2D footprints which are suitable for 2D map applications for achieving a better rendering performance. For example, many mapping applications utilize the 2D map engine i.e. OpenLayers to establish the cadastral maps which allow public users to explore and interact with the city objects such roads, buildings, and addresses etc. with high viewing performance.

- ***Extruded***: This display form is a generalized 3D representation of the original city objects. It results from an extrusion of the *Footprint* geometry to their height according to the height value of the respective 3D object. The height value can be achieved by two ways: The first way is to query the value of the attribute “Measure\_Height” from the database. However, this attribute is only available for the CityGML class “Building” but not for other feature classes. Besides, this attribute can have a NULL value since it is not a mandatory building attribute according to the CityGML standard. Another way is to calculate the height value of the object’s 3D bounding box which can be queried from the ENVELOPE attribute of the respective city object from the database. This is a very generic solution since the ENVELOPE attribute is defined for all city objects and can also be automatically calculated from the geometry information of the city object if the ENVELOPE attribute has a NULL value.
- ***Geometry only***: with this display form, all city objects are represented with fully detailed geometry information in accordance with the CityGML’s Level of Detail. For example, when exporting an LOD2 building object, all its thematic surfaces such as ground surfaces, wall surfaces, and roof surfaces etc. can be exported and explicitly displayed on the 3D map. To achieve a better rendering performance, the ground surfaces can be omitted during the exports since they are typically hidden by the above wall and roof surfaces and cannot appear on the 3D map anyway. In addition, for improving the viewing effect, it is possible to assign different colors to the individual thematic surface types, for example roof with red color, and wall with grey color. These colors can be further tuned by using the alpha value which affects the transparency of the surface appearance. In case that the thematic surfaces are not explicitly modeled in the data model, the wall and roof surfaces can be distinguished by following a trivial logic according to which surfaces touching the ground will be considered wall surfaces, all others will be considered roof surfaces. However, this approach is only applicable for LoD1 and LoD2 models because many additional building constructions like windows, doors, and installations can exist in higher LoDs (LoD3 and LoD4) which will violate this rule.
- ***Geometry with Textures***: Compared to the display form “Geometry only”, this display form shows much more sophisticated information regarding the surface appearance such as materials and textures. Thus, this display form can provide rich visual information for the realistic 3D representation of the city object on the 3D map. For example, the texture images of the building surfaces can be captured from orthophotos or street views and attached to the surfaces of the city objects by using the Appearance module defined in the CityGML standard. These appearance information can be directly used for generating textured 3D visualization models which are supported by most 3D graphic formats like COLLADA and glTF. In addition, for the same city

object, CityGML allows to define multiple appearance themes simultaneously which can be used to generate different 3D visualization models for the same object.

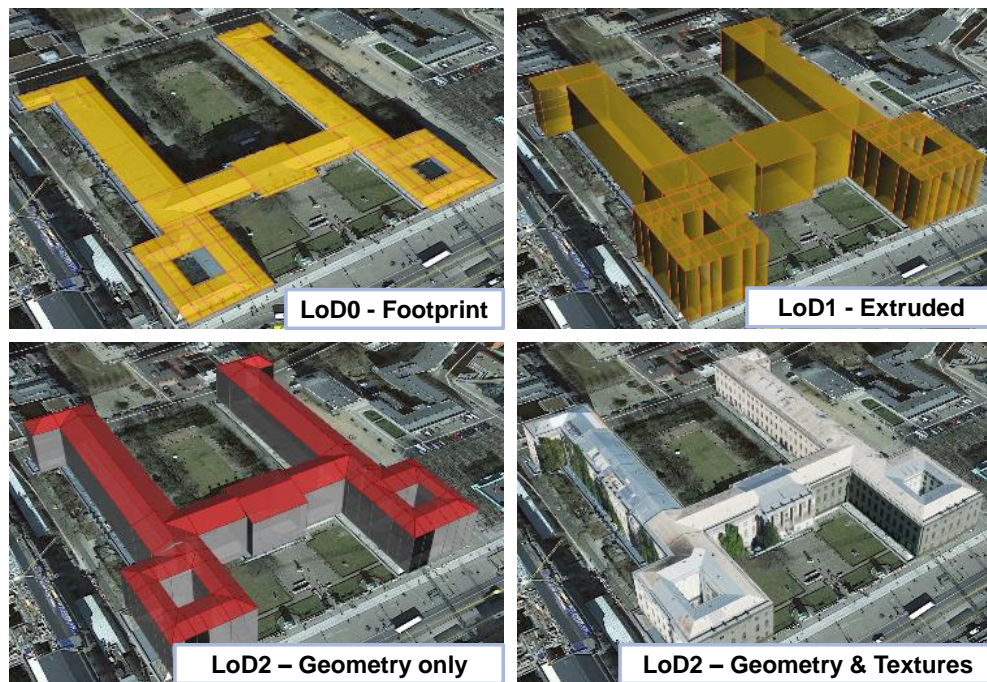


Figure 77: Example of different display forms of the created 3D visualization models

While creating 3D models whose surfaces are rendered with a constant color, it is important to calculate surface normal value for each surface during the exports and add the calculated value to the 3D mesh. The reason is that the surface normal allows the object surfaces to be illuminated with a shading effect in a 3D scene and therefore provides a better visual representation. If the surface normal is not calculated, the 3D models will be rendered as a solid geometry without any visual distinction of its boundary surfaces (cf. Figure 78). However, when exporting textured 3D models, the shading effect is not crucial, since the texture information can already provide a sophisticated visual effect through the surface appearance e.g. textures.

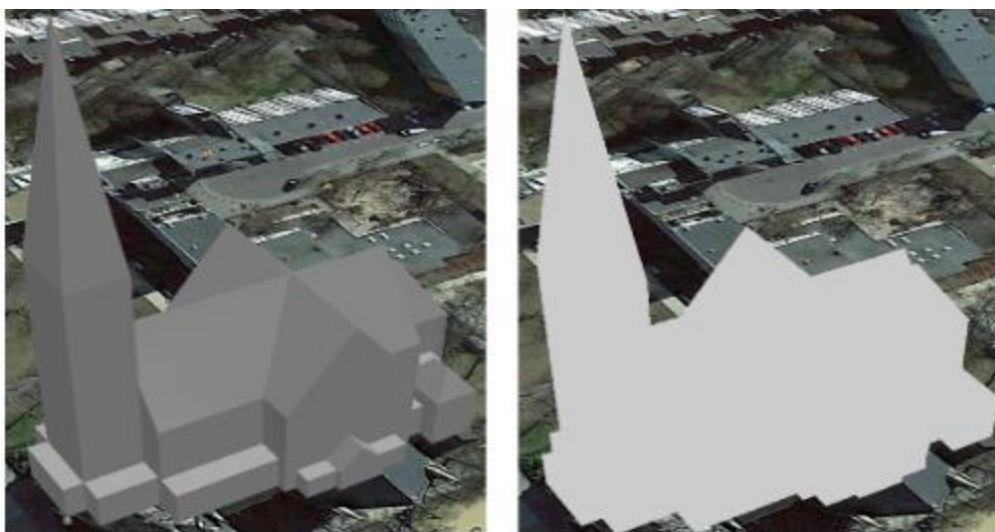


Figure 78: Comparison of the different visual effects of the same 3D model with (the left figure) and without (the right figure) surface normal

When creating 3D visualization models with textures, there are a number of important aspects that must be considered regarding the visual effects and performance issues. In CityGML models, multiple texture images can be clustered together and stored in one image file by grouping them into a very large canvas called ‘texture atlas’ which allows to increase the overall storage efficiency for hard drive and rendering performance for 3D viewer applications. However, in some CityGML datasets, it might occur that a very large texture atlas image is shared by multiple surface geometries belonging to many different city objects. In this case, every exported 3D graphics model representing a city object will receive a complete copy of the texture atlas image in which only a small portion of it is actually used. This will however result in poor rendering performance when loading and rendering the individual 3D object. In order to overcome this issue, the original texture atlas image can be cropped into a number of small texture images, each of which should correspond to one surface geometry and could be very small in size. In the subsequent step, the split surface images belonging to the same city object will be joined together again to create a new texture atlas. In this way, an optimal rendering performance can be achieved when viewing the individual city object (cf. Wloka 2005). In case that more than one city object are joined together to form a batched model which will be rendered in a draw call, a larger atlas for this batched model shall be created.

For creating a compact texture atlas, a number of algorithms have been developed which generally solve the problem of the two-dimensional image packing, also known as ‘knapsack problem’ (cf. Coffman et al. 1980). One of the simplest algorithms is using a binary tree structure to subdivide the atlas bin into a set of small rectangles which can be used for storing the texture images. In the first step, an empty atlas bin will be initialized, and the first item will be added and placed at the top-left corner of the bin. In the next step, the remaining empty region is split into two rectangles along the larger side of the last added item. The next item is inserted into one of the free rectangles and the remaining empty space is split again. The items can be rotated when being inserted into the texture atlas in order to best fit the size of the available empty space. Performing these steps in a recursive way can build a binary tree structure to represent the texture atlas. An example illustrating this algorithm is shown in Figure 79.

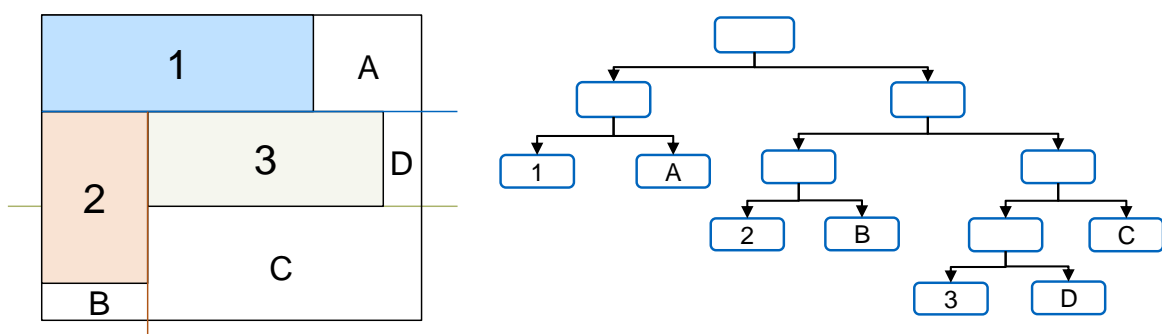


Figure 79: Example of the first packing algorithm

Another packing algorithm is called Touching Perimeter (TPIM) algorithms which is a heuristic approach allowing to generate more efficient texture atlas with higher area/total atlas size ratio (cf. Lodi et al. 1999, Lodi et al. 2002). TPIM also initializes a bin with a maximum acceptable size and packs one item at one time. All items going to be inserted shall be sorted according to descending area values. The first inserted item is placed at the bottom-left



corner. Each following item is packed with its lower edge touching either the bottom of the atlas or the top edge of another item, and with its left edge touching either the left edge of the atlas or the right edge of another item. The packing position should be chosen by evaluating a score value. This value is defined as the percentage of the item perimeter which touches the atlas borders and the other items that have already been inserted. For each candidate item, the score will be calculated twice according to the original orientation and the one with 90 degree rotation. The orientation with the highest score value will be taken, and the candidate item will be inserted accordingly. A comparison of results of the two mentioned algorithms are displayed in Figure 80. Also, there exist many other packing algorithms such as SLEA, NFDH, and NFDH etc. (cf. Rode & Rosenberg 1987)

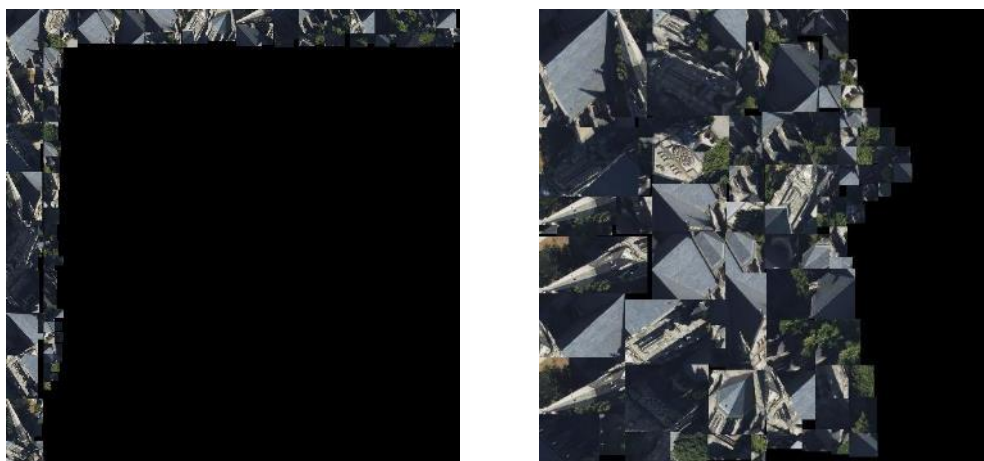


Figure 80: Comparison of the generated texture atlas images created using the Basic (left) and TPIM (right) packing algorithms

Scaling texture images is another important way for reducing file size to increase the loading and rendering speed. This can be done by multiplying the image sizes with a scale factor ranging from 0 to 1, where 1 means no scaling. From the practical experience, a test 3D model with different sample scale factors such as 0.2, 0.5 and 0.7 etc. shall be first generated and visually examined to select the proper one that should offer a fairly good image quality and also has a major positive effect on the viewing performance. Therefore, the general workflow for creating a textured model is to first generate texture atlases and then scale them to a smaller size.

### 5.1.2 Tiling of 3D Visualization Models

In the practical application cases, the file size of the generated 3D visualization models of large city regions can easily exceed several gigabytes or even more. Loading and displaying all these 3D visualization models at one time will rapidly exhaust the CPU, GPU, and memory resources of most user computers. To handle this issue, all the exported 3D visualization models must be organized in a certain spatial structure by dividing the exported 3D model objects into a number of small tiles at the server side (cf. Christen 2016). The meta-information i.e. the file location or the minimal 3D bounding box of each tile shall be encoded in a separate file which shall have a very small file size such that it can be easily parsed by the 3D viewer application. Based on this meta-information, the application is able to determine the relevant data tiles according to the current camera perspective and load the corresponding

3D visualization models from the server. The conceptual idea of such meta-information about the tiling structure is called “Tiling Schema”.

A well-known tiling schema is the Tile Map Service (TMS) Schema which is based on a quad-tree structure to split the entire geospatial data into a number of hierarchical levels (cf. Tile Map Service Specification 2017). A total number of 27 levels are specified which range from level 0 representing the lowest level up to level 26 being the highest level. At the root level 0, two tiles are initially defined which can be recursively split up into four children tiles down to the lowest level. All tiles belonging to the same level are evenly subdivided in the longitude and latitude orientation (cf. Figure 81). In this way, each tile can be seen as a container for holding the data lying within it to build a pyramid hierarchy based on which the data with lower resolutions can be stored in the tiles at lower levels whereas the data with higher resolutions are located at the tiles with higher levels. With this approach, a mapping application is able to dynamically load and display the respective data tiles according to the extent of the view at runtime. For example, when the map is zoomed in, the tiles at higher levels shall be loaded to replace the tiles at the lower levels for showing the detailed information. Due to its simple spatial structure, the TMS tiling schema has been widely used for the implementation of OGC Web Map Service and Tiling Map Service etc. for streaming imagery and raster data covering the entire earth surface. However, the main drawback of this tiling schema is that it has a fixed tiling structure and lacks the ability to allow users to structure their data based on a customized tile size or spatial structure.

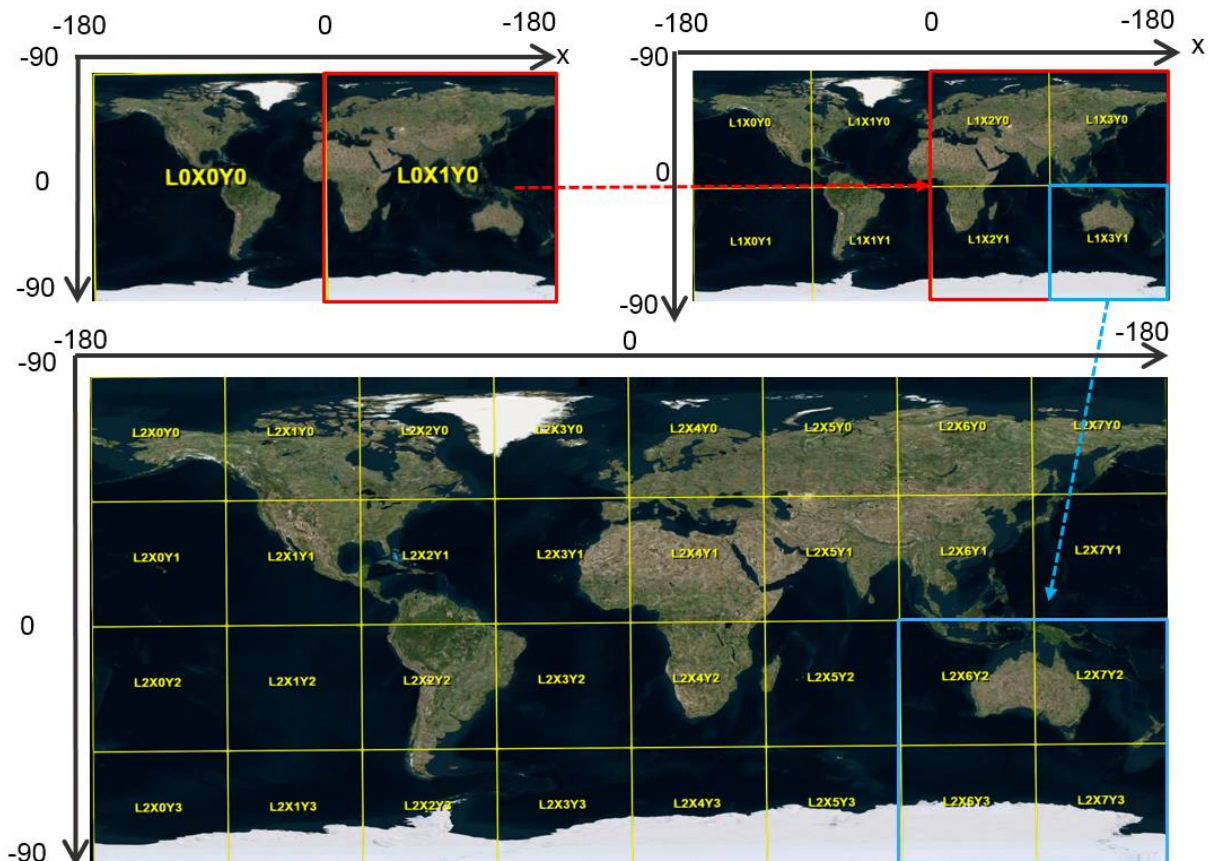


Figure 81: Excerpt of the TMS layout based on the Web Mercator Auxiliary Sphere

Compared to the TMS tiling schema, a more powerful and generic tiling schema is available called 3D-Tiles which is a general specification for efficiently streaming and rendering

heterogeneous geospatial data on the web (cf. Cozzi 2017). It supports the 3D geospatial contents with different formats such as 3D models, digital terrain model, and point cloud etc. The 3D-Tiles specification has been initially designed by the company AGI who developed the Cesium Virtual Globe and is now released as an OGC standard (cf. Cozzi et al. 2019). During the past two years (2015 - 2017), more and more software vendors from the 3D GIS domain employ the 3D-Tiles standard to deploy their 3D geospatial data for the high-performance rendering and 3D visualization in the Cesium virtual globe.

According to the 3D-Tiles specification, the 3D geospatial data can be tiled into an arbitrary number of data tiles which can be organized with a variety kinds of hierarchical spatial structures such as k-d trees, uniform and non-uniform quad-tree, and octrees (cf. Figure 82). The tiles can overlap with each other but should preserve the spatial coherence which means that the spatial content of child tiles shall be completely inside the spatial extent of the parent tile. Therefore, the 3D-Tiles standard provides a more flexible way for defining the tile structure according to the specific application needs or data characteristics. For example, when handling massive 3D building objects, the quad-tree structure is very suitable, since the buildings in a city are evenly distributed on the 2D planar dimension. An Octree structure is often chosen for handling massive point clouds that may have a large density in a 3D extent or to represent a building object with a higher level of detail.

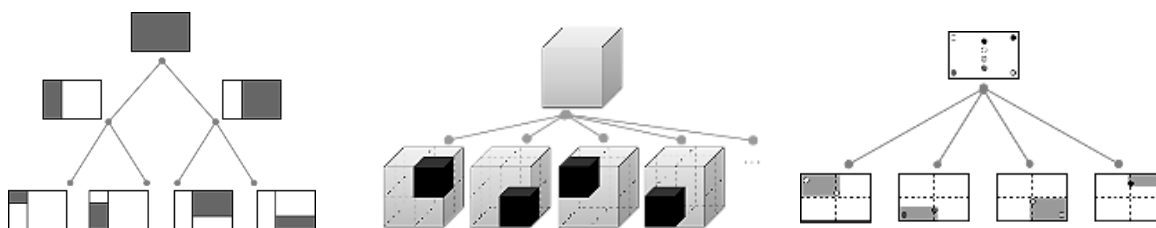


Figure 82: Typical spatial data structure supported by the 3D-Tiles standard (cf. Cozzi et al. 2019)

The hierarchical spatial structure of the 3D-Tiles datasets is described using a set of small JSON-formatted files which are used for storing the metadata of the generated data tiles. In each metadata file, a set of nested JSON objects are encoded each of which is called *tileset*. Each *tileset* object receives a couple of metadata parameters for describing the relevant meta-information of each tile. The first one is called *boundingVolume* to define the spatial extent of the corresponding tile. This bounding volume can be a 3D bounding box or a 3D sphere which will be used for performing the view-frustum culling calculation to determine if the tile should be loaded or not. Further, the next parameter *geometricError* is a nonnegative number to determine, if the respective tile can be actually visualized or not, even if it lies within the view frustum. If the *Screen-Space Error* (SSE) of the current camera view is larger than the *geometricError* value, then this tile shall be invisible and replaced by the parent tile with coarse resolution. The next relevant parameter is *content* which store the path or URL of the actual data contents or another *tileset* file. The last parameter is *children* which is an array of *tileset* objects for representing the children tiles. Using this nested structure, applications are able to tranverse the metadata file to fetch the corresponding data tiles according to the camera perspective. The 3D-Tiles strategy offers a powerful means for organizing and streaming geospatial data but also increases the complexity of the data generation process. Therefore, it is difficult for most software vendors to manage the creation of 3D-Tiles

compliant 3D models, and to the date of writing the thesis, there exist no open-source solutions supporting the full automatic generation of 3D-Tiles data from the semantic 3D city models i.e. CityGML do exist yet.

Compared to the 3D-Tiles approach, a lightweight tiling schema has been developed in the context of the thesis. It provides a simple tiling schema which was developed even one year before the 3D-Tiles came up. This tiling approach is based on a simple grid-based structure for the tiling of the geospatial data according to a chosen bounding box and a tile size (side length). When performing the data tiling process, the amount as well as the extent of the tile rows and columns will be calculated by dividing the length and width of the bounding box. The 3D model objects i.e. buildings whose centroids lie within each tile's bounding box will be assigned to the tiles and can be encoded in different 3D visualization formats i.e. KML, COLLADA, glTF etc. In this way, all data tiles will be generated with a hierarchical directory structure where each individual tile file shall be named with its column number and all the tile files that belongs to the same row are stored in a separate subfolder named with their corresponding row number. The numbering of both rows and columns should start with 0. All those subfolders are in turn stored in a folder named “Tiles”. An example shown in Figure 83 indicates that this hierarchical directory structure can clearly reflect the underlying grid-based spatial data structure and also ensures that the exported tile files are distributed over different subfolders in order to avoid putting all tile files into a single folder which may result in significant performance issues at least under the MS Windows operating systems.

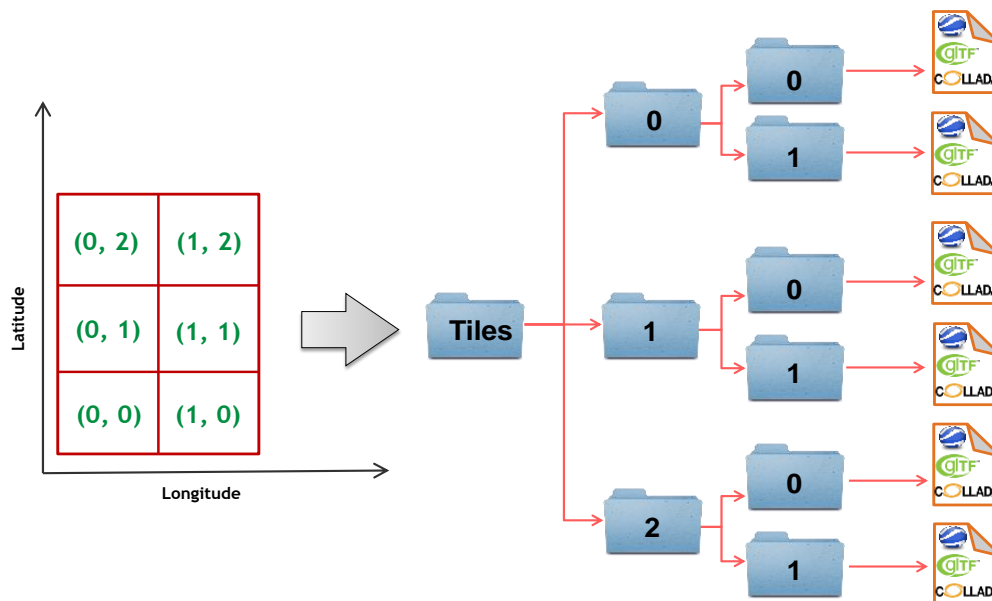


Figure 83: Hierarchical directory structure for export of 2x3 tiles using the grid-based tiling layout

Additionally, the number of rows and columns can also be freely specified by users. In this case, the global bounding box will be divided into equally spaced portions horizontally and vertically based on the WGS84 geographic coordinate reference system. The tile size will hence be automatically adapted according to the spatial subdivision. This tiling option could be appreciated in case that the users want the amount of the generated tiles to not exceed a certain limit. However, it is not possible to ensure that the tile side length to be equal in the both horizontal and vertical directions. Therefore, a non-square tile i.e. may be generated if the global bounding is very long and narrow, and the specified row and column numbers are



equal. Such data tile form is not sufficient for the visualization, because when a tile is intersected with the view frustum, a number of 3D models lying outside the view frustum may also be loaded and slow down the viewing performance.

The meta-information of the generated data tiles is stored in a JSON file which has a very simple structure and can be easily interpreted by applications. An example is shown in the following.

#### Meta-Information of the grid-based Tiling:

```
{
  "layername": "NYC_Buildings",
  "fileextension": ".kmz",
  "displayform": "extruded",
  "colnum": 29,
  "rownum": 23,
  "bbox": {
    "xmin": -74.0209007,
    "xmax": -73.9707756,
    "ymin": 40.6996416,
    "ymax": 40.7295678
  }
}
```

As the name implies, the first parameter *layername* denotes the name of the generated datasets. It is also used as the prefix of the file name for each data tile. The value of the *fileextension* can be “.kml”, “.kmz”, “.collada”, or “.glTF” to denote the format of the generated 3D visualization models. The next parameter *displayform* corresponds to the four display forms mentioned in the previous subsection. Using these three parameters, the name of a tile data can be constructed as follows:

$$[\text{layername}] + [\text{row}] + [\text{col}] + [\text{displayform}] . [\text{fileextension}]$$

With this naming rule, if the row and column grid coordinates of a tile is given, applications can quickly find the target data tile from the server according to the tiled file structure and the file name.

The parameters *colnum* and *rownum* correspond to the amount of the generated tiles whose global bounding box is represented using the parameter *bbox*. With the combined use of these three parameters, applications are able to perform a simple spatial calculation for searching those data tiles that are intersected with a given spatial point.

For example, the length and width (in WGS84) of each tile can be determined using the following formulas:

$$\text{TileWidth} = (\text{bbox.xmax} - \text{bbox.xmin}) / \text{colnum}$$

$$\text{TileLength} = (\text{bbox.ymax} - \text{bbox.ymin}) / \text{rownum}$$

With these two calculated values, applications are also able to use the following formulas to rapidly retrieve the row and column number of the tile in which a given point lies:

$$\text{ColumnNumber} = \text{floor}((X - \text{bbox.xmin}) / \text{TileWidth})$$

$$\text{RowNumber} = \text{floor}((Y - \text{bbox.ymin}) / \text{TileLength})$$

where *X* and *Y* denote the WGS84 coordinates of the given point.

Further, if a bounding box is given, which is formed by a lower-left corner and an upper-right corner and their row and column numbers are expressed as  $(R1, C1)$  and  $(R2, C2)$  respectively, all those tiles that intersect with the given bounding box can be found iteratively, as their row and column numbers must fulfil the following conditions:

$$R1 \leq RowNumber \leq R2 \wedge C1 \leq columnNumber \leq C2.$$

For example, while the 3D map is being navigated, the screen of the current camera view shall be first projected onto the earth surface in order to build a quadrangle whose outlines are painted with red color as shown in Figure 84. In the subsequent step, a maximum bounding box (painted with blue color) enclosing the projected quadrangle shall be automatically calculated by the applications and will be used for finding out all those tiles that are intersected with the camera view or very near to the camera position. In this example, the tiles whose row and column coordinates are in the range of  $[1, 5]$  and  $[1, 6]$  are matched. These tiles are handled as the so-called candidate tiles which will be further processed and analyzed by the map applications to determine, whether the tile should be loaded for the visualization or not.

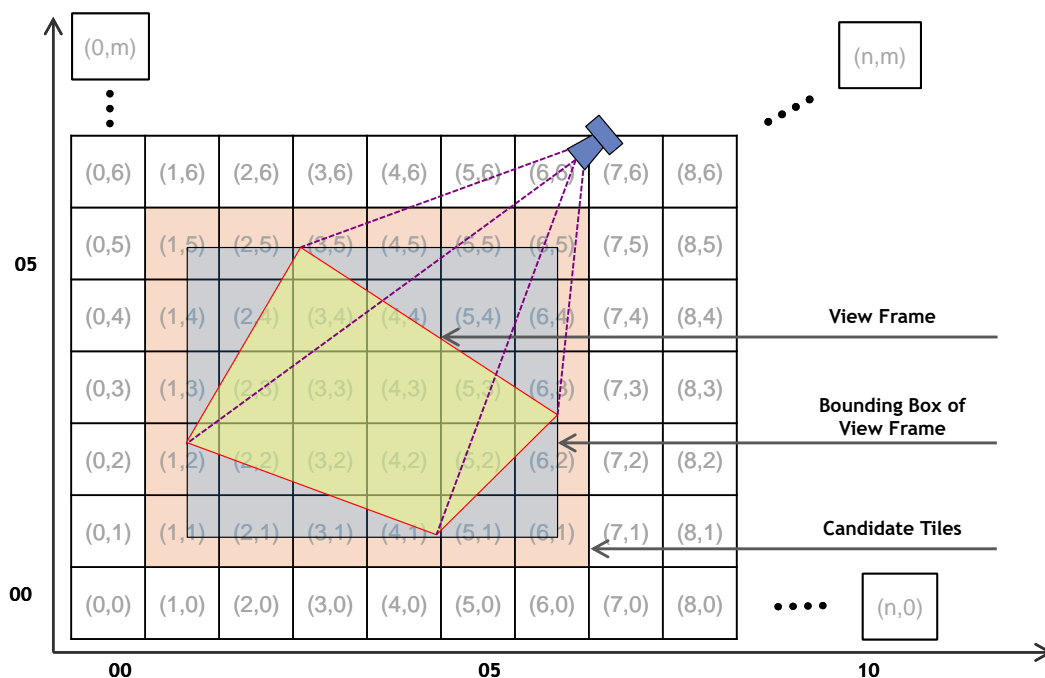


Figure 84: Strategy for determining the candidate data tiles that should be loaded according to the camera perspective projected onto the screen space

### 5.1.3 Streaming and Visualization of Tiled 3D Visualization Models

In 3D map applications, the dynamic streaming and visualization of 3D visualization models is generally based on the so-called Level of Details (LoD) concept according to which the data tiles with higher resolution should be displayed when the observer is viewing them from a short distance. When data tiles are far away from the observer, the data tiles with higher resolution should be substituted by the data tiles with lower resolution. Such LOD concept can be easily done by means of the simple and lightweight tiling schema introduced in the previous section. Basically, the determination of visible data tiles is realized by means of the combination of the two parameters *minLodPixels* and *maxLodPixels* which define the

minimum and maximum limit of the visibility range for each tiled dataset. The allowable value range starts at 0 and ends at an infinite value expressed as -1. In the map application, all the candidate tiles (cf. the previous subsection) are projected onto the screen space (cf. Figure 85). During the runtime, the maximum diagonal length of each tile shall be calculated and measured in screen pixel. If the diagonal length is larger than *minLodPixels* and less than *maxLodPixels*, then the respective data tile shall be loaded and displayed. Otherwise it should be hidden from display. In addition, all data tiles lying outside of the view frustum must not be further processed and should be unloaded and invisible anyway. The approach of using *minLodPixels* and *maxLodPixels* originates from the KML specification, which provides a so-called “NetworkLink” format for streaming and visualizing tiled 3D KML data models in the Google Earth 3D viewer. This approach has been further enhanced by means of the developed tiling schema to support the efficient 3D model visualization on the open-source Cesium Virtual Globe.

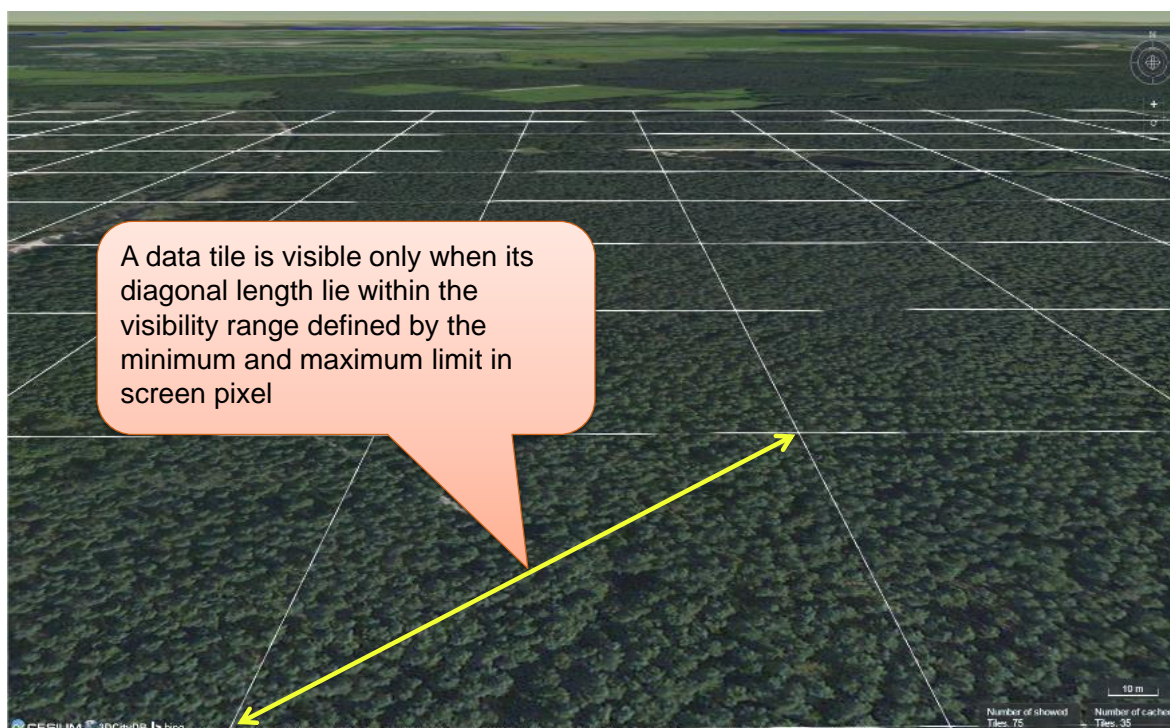


Figure 85: Efficient determination of which data tiles should be loaded according to the user-defined visibility range in screen pixel

When viewing from a long distance to the Earth ground, if the visibility range specified by the user starts at a very small value and ends at an infinite size, massive amounts of data tiles will tend to be loaded which can also result in poor performance of the 3D rendering or even memory overload of the web browser. In this case, each data tile will always be visualized even though it only takes up a very small screen space. This issue can be avoided by a proper setting of an additional parameter called *maxNumberOfVisibleTiles* which specifies the maximum number of allowed visible data tiles. When this limit is reached, any additional data tiles that are farthest away from the camera will not be shown, regardless the size of screen space they occupy. According to many practical tests, this value can be set to 200 if the tile size has been defined as 150\*150 meter, which is an appropriate tile size in most use cases. Of course, depending on data volume of each tile and the capability of the hardware such as

main memory size being used, this parameter value can be adjusted to archive a better viewing quality.

Moreover, the map application can be further optimized by implementing a so-called caching mechanism (cf. Figure 86) which allows high-speed reloading of those data tiles that have been loaded before and temporarily stored in the memory of the web browser. In order to prevent memory overload, an additional parameter *maxSizeOfCachedTiles* can be used for specifying the maximum allowable cache size expressed as a positive integer number. With this parameter, the map application is able to implement the so-called Least Recently Used (LRU) algorithm which is a caching strategy being widely used in many applications systems for the cache control management. According to this algorithm, the newly loaded data tiles will be successively put into the cache and sorted in a queue list which can be managed in a background thread running in parallel. When the cache size limit is reached, the map application will remove the least recently visualized data tiles from the cache. Obviously, the value of this parameter has to be equal or higher than the value of the parameter *maxNumberOfVisibleTiles* and can be increased in order to achieve a better viewing experience depending on the capability of the hardware being used. The caching process runs in parallel to the rendering procedure until the camera stops moving and the candidate data tiles have been successfully rendered.

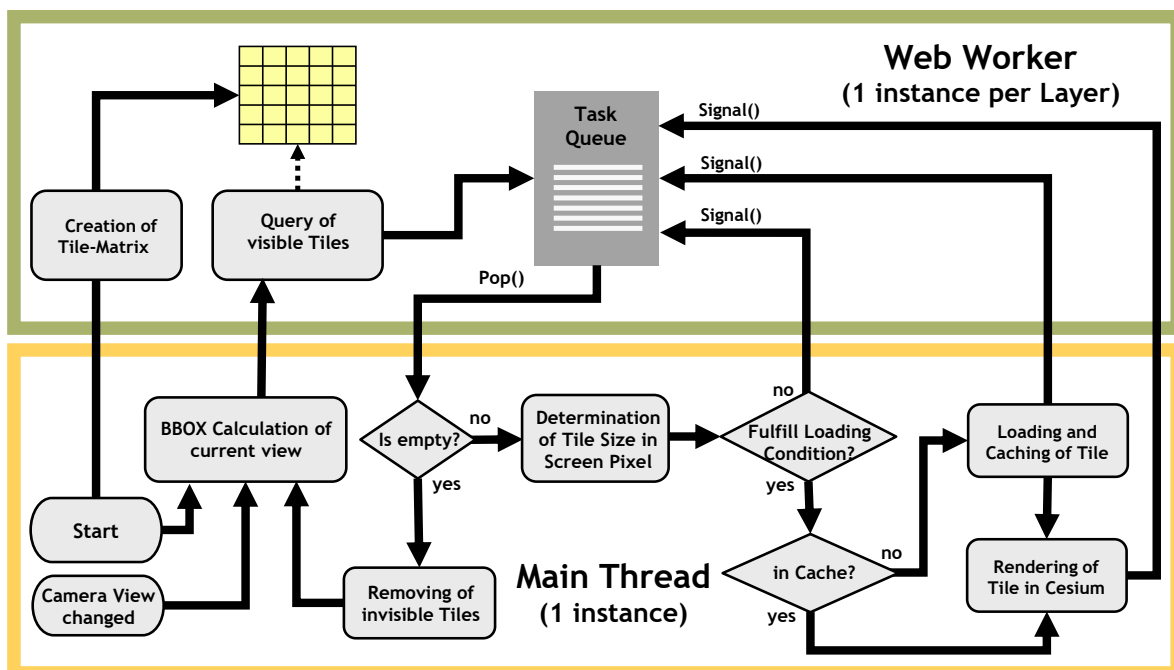


Figure 86: Workflow of performing the determination and caching of the data tiles for efficient 3D visualization

## 5.2 Exploration of Semantic 3D City Models

Apart from the efficient 3D visualization, the data exploration of the individual or a group of 3D objects is also an essential functionality to enhance a 3D map application allowing to work with semantically rich 3D city models for accomplishing various application tasks. To realize this, two important aspects must be considered which include the interaction of the 3D objects as well as the linking with dynamically extendable and domain-extendable attribute information during the runtime for enriching the 3D visualization models. All these



functionalities can be realized using a web-based 3D map application which is able to serve as an extensive platform for testing the developed functionalities on the one hand, on the other hand, it can also be employed in many research and practical projects in the context of “*Smart Cities*”.

### 5.2.1 Interaction with 3D City Model Objects

The rich model interaction requires the highlighting of the 3D objects on mouse over and mouse click. Ctrl-clicking mouse operation should be able to select or deselect more than one 3D object interactively. Once a building has been selected, it shall be automatically highlighted, and its attribute information shall be displayed next to the selected 3D object. The attributes and their values can be carried along with the spatial contents in the 3D visualization model or queried from a remote server through a standard web accessing API e.g. Web Feature Service or Cloud services (cf. the next subsection). Additionally, it is also important that the selected objects can be interactively hidden and redisplayed in the 3D map application. This functionality is especially useful for simulating different urban planning scenarios and architectural designs in the decision-making process. For example, an urban planner or an architect may want to hide an existing building object and insert a new one for editing and comparing different 3D designs of the same building. The different planning scenarios can be shared with other project partners to allow for collaborative work on the same workspace. In addition, a 3D object can be visually inspected in different kinds of map viewers such as street view, bird-eye view, 2D top-down view or a combined version (cf. Figure 87) for checking and validating a virtual 3D model against the real one captured from the real scene.

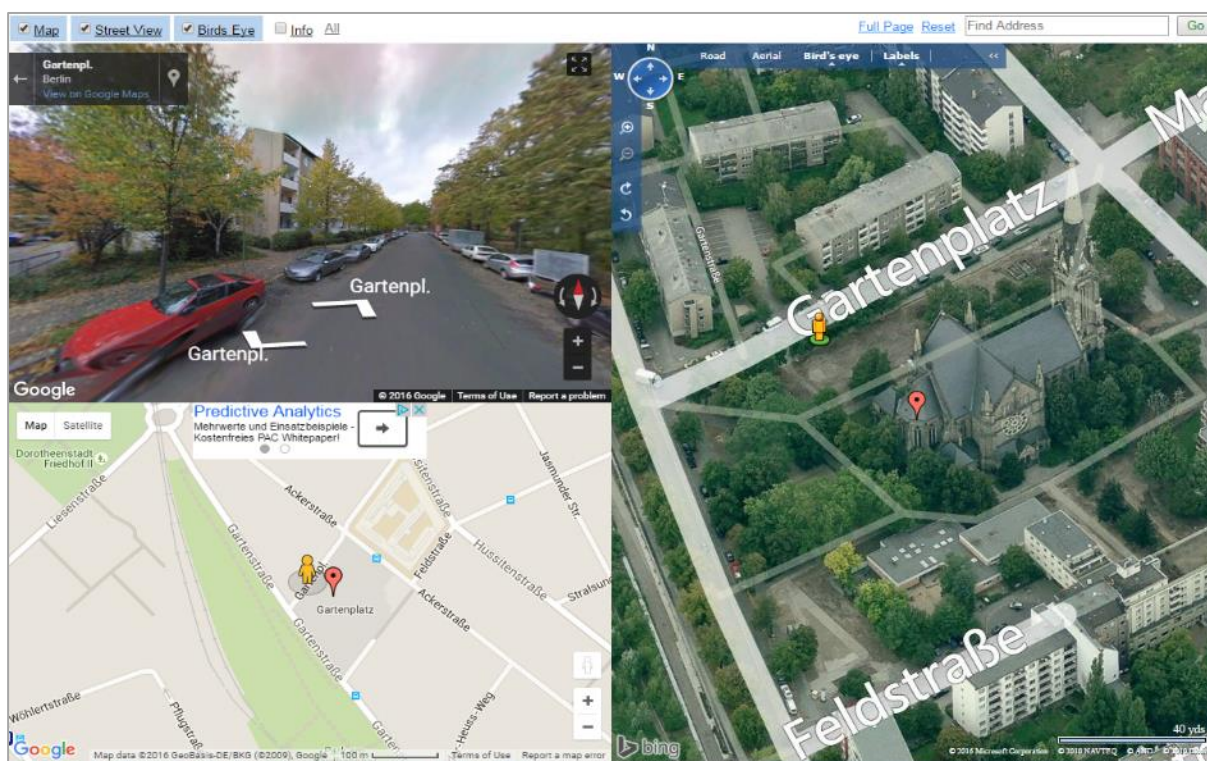


Figure 87: Exploration of a building in a mash-up view using ‘Dual Maps’ ([www.dualmaps.com](http://www.dualmaps.com))

The interaction with deeply structured semantic 3D city models is another important problem for the 3D model interaction. CityGML comprise the spatial and graphical aspects of the city objects along with the ontological structure including thematic classes, attributes, and their interrelationships. Objects are decomposed into parts on the basis of deeply nested structures that can be observed in the real world. For example, a building will be decomposed into several building parts which can again consist of parts like wall and roof surfaces. Since CityGML objects can have attributes and relations on all levels of this aggregation hierarchy, the interaction with such deeply nested structures of the 3D city objects must be taken into account.

This objective can be reached by using a so-called batched 3D models (B3DM) which is a glTF-based format for transferring and rendering a group of 3D objects within a single file. In a B3DM file, each model object has a unique identifier being assigned to every respective vertex by means of its *batchId* attribute which is an additional attribute to the 3D coordinate information. In addition, the B3DM allows to contain an extendable information table called *Batch Table* which can be used for storing arbitrary attributes of the batched 3D model objects and which is also indexable by using the *batchId* attribute of the vertex. With the help of the *Batch Table*, the semantical feature hierarchy can be fully described. An example is shown in Figure 88.

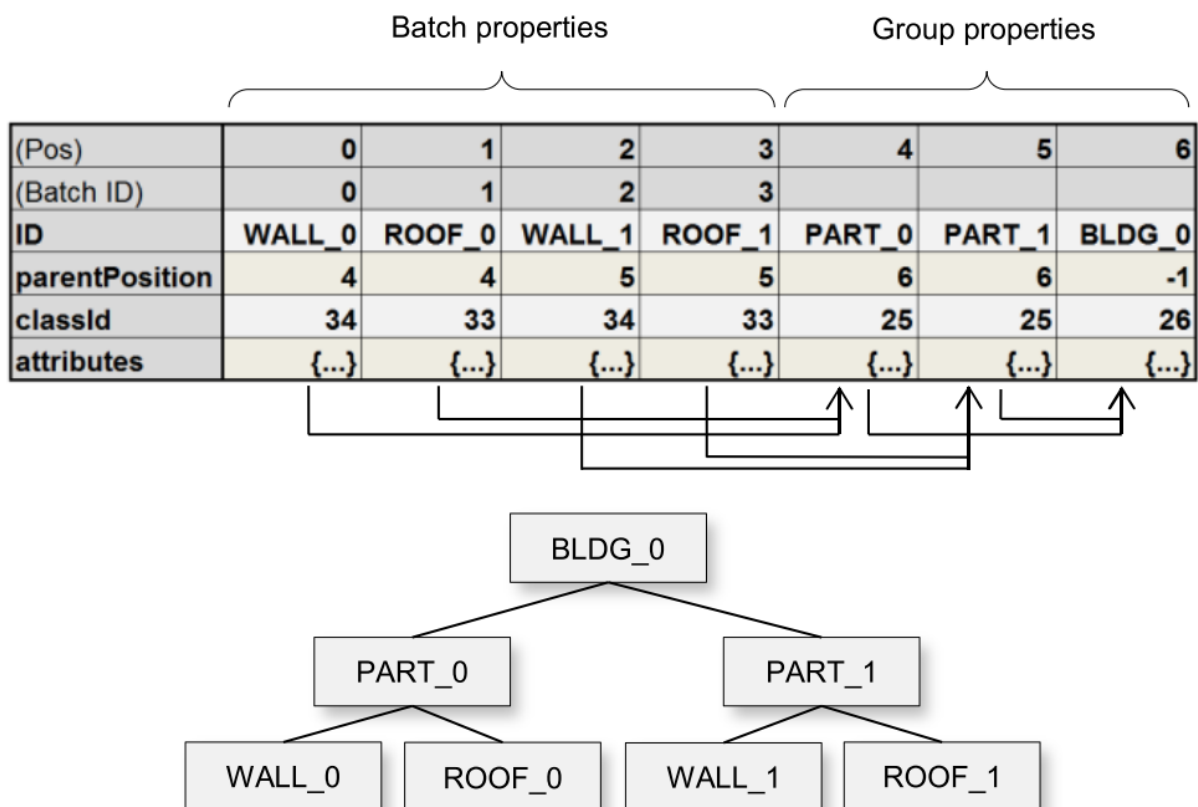


Figure 88: Modelling of the hierarchical structure of 3D city models using the glTF-based B3DM format (cf. Schilling et al. 2016)

This batch table can be horizontally subdivided into two parts. The first part called *batch properties* holds all the primitive objects of an aggregation hierarchy. The second part called *Group properties* represents those objects that are not explicitly constructed by vertex but are aggregated by the primitive object stored in the first table part. The hierarchical relationship

between the aggregating and aggregated objects are represented using the *parentPosition* attribute. The attribute value -1 indicates that the respective object is the root object of the aggregation hierarchy. The attribute *classId* is used for determining the CityGML class type of the respective model object whose actual attributes information are stored in the attribute *attributes* encoded as a JSON structure. In this way, the attributes as well as the structure information of a complex 3D city object with multiple aggregation levels can be fully stored within the 3D visualization model which can be easily explored by the users using a map application.

### 5.2.2 Coupling of 3D Visualization Models with Thematic Information

A major problem regarding the exploration of the 3D visualization models is the linking with and exploration of thematic information. Although the 3D models like KML and B3DM allow to store thematic information together with their spatial contents, the dynamic augmentation and enrichment of thematic information is computationally very expensive, since the entire 3D visualization model will have to be regenerated and uploaded to the server again. This problem can be solved by using a Web-based API e.g. Web Feature Service (WFS) which allows applications to dynamically query the thematic as well as the spatial information from a database instance based on a selected object ID (cf. Figure 89). However, this approach also has its own drawbacks. First, the setup of a WFS server is a time-costly work which requires a number of configurations for binding the database server with the WFS application server. Second, if users want to add their own attributes information, the data contents in the database must be augmented accordingly. However, this is often impossible since the central database should be maintained as a stable data repository which shall prevent normal users from changing and modifying the table contents.

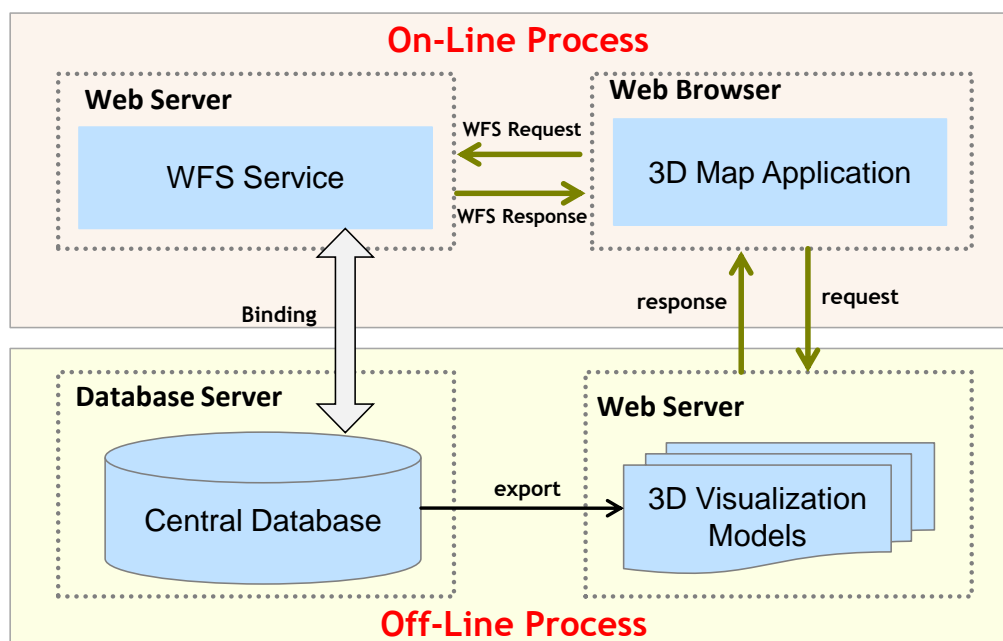


Figure 89: Example system architecture of coupling WFS with 3D visualization models in a 3D web map application

A complementary solution is to utilize a Cloud-based online spreadsheet for storing the thematic information and any update of thematic contents can exclusively take place within

the online spreadsheet and therefore does not require exporting and deploying the 3D visualization models again. The basic idea of linking online spreadsheet with 3D visualization models is illustrated in Figure 90. Like with the structure of a typical database table, the first row of the online spreadsheet defines the attribute names, and the further rows store the respective attribute values for each 3D object. The logical links between the 3D models and the respective rows are established via a specific column within the spreadsheet, namely the GMLID column which lists the unique identifiers of the 3D objects. Each further column is used to represent one attribute of the 3D object. Since most cloud services i.e. Google Drive and Microsoft OneDrive etc. usually provide comprehensive REST APIs allowing to query the content information from an online spreadsheet via a HTTP GET/POST request, the ad-hoc query of thematic information of a selected city object is possible within a 3D map application. In addition, since the Cloud services also supports the handling of the access control, the owner of an online spreadsheet can also share it with other users who will also be able to access the online spreadsheet. This enables multiple users to work on the same online spreadsheet in a collaborative way. Furthermore, the Cloud services usually offer a web portal allowing users to alter the online spreadsheet by modifying the existing attribute values or inserting a new attribute by means of adding a new spreadsheet column. In this way, the online spreadsheet can be used as an intermediate data container for keeping the contents up-to-date without affecting the original (possibly official) 3D city model stored in the central database.

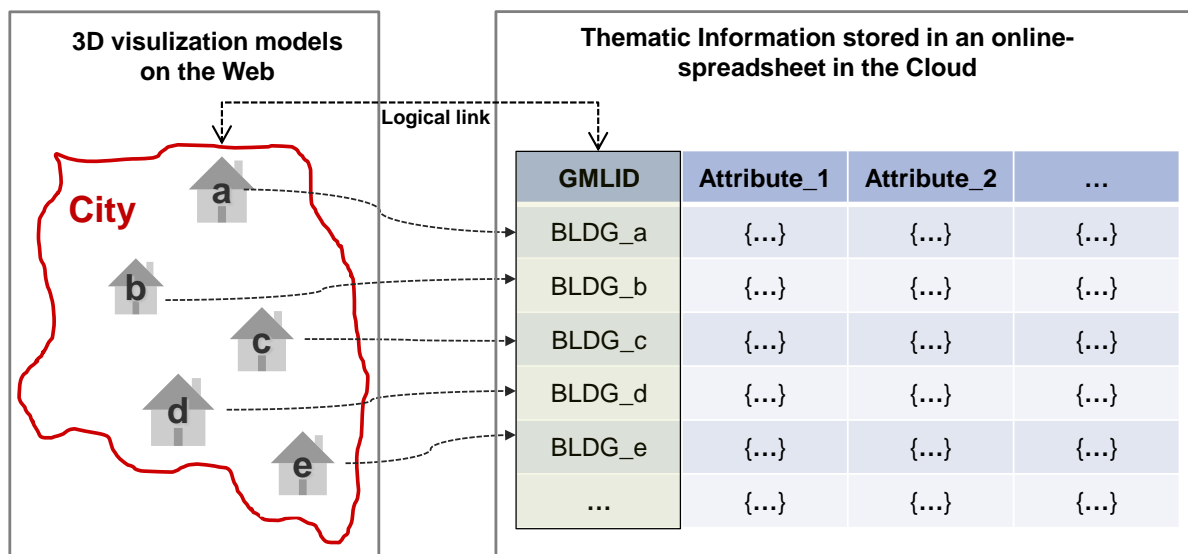


Figure 90: Coupling an online spreadsheet with a 3D visualization model via GMLID

The typical system architecture of the coupling of Cloud-based online spreadsheets and 3D visualization models is illustrated in Figure 91. In analogy to the WFS-based application environment, the 3D visualization models shall be first generated through an off-line pre-processing step and uploaded to a web server. In another off-line processing step, the thematic information can be exported from the central database and uploaded to an online spreadsheet where the first column is per-default the GMLID column which lists all the object IDs of the exported city objects in the 3D visualization model. Using this GMLID column, the online spreadsheet can be expanded by joining with another spreadsheet which can contain other domain-specific or user-defined attribute information of the exported city objects. While



exploring the 3D visualization model in the 3D map application, users can visually inspect the individual city object rendered on the 3D map and update the content information in the online spreadsheet accordingly. In the last step, which takes place off-line, the edited and enriched content information can be checked and validated by application administrators and the valid data contents will be written back to the central database to make the 3D city models up-to-date.

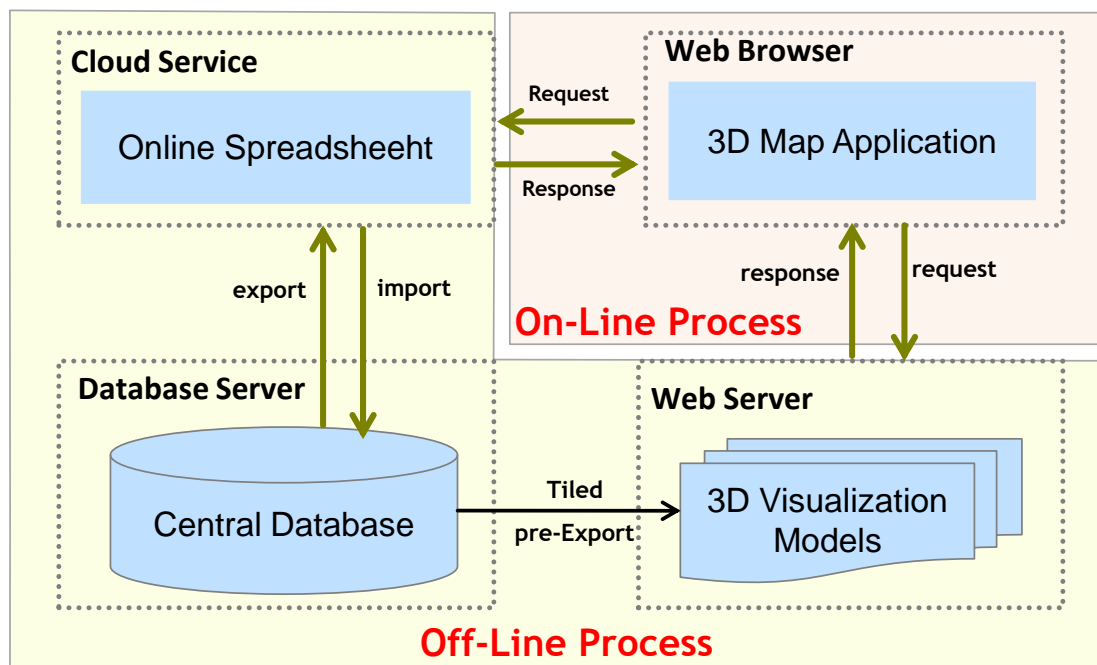


Figure 91: Example system architecture of coupling online spreadsheet with 3D visualization models in web applications

In practical applications, a mash-up of and an M-N relationship between different 3D visualization models and online spreadsheets are usually needed, rather than the simple 1:1 relationship. On the one hand, a city object, whose attributes are stored in an online spreadsheet, may occur in different 3D visualization models exported with different display forms e.g. footprint, extruded, or textured geometries. On the other hand, a 3D visualization model representing a data layer can be linked with more than one online spreadsheet which can be individually used for storing different categories of thematic information belonging to different domains. For instance, one spreadsheet can be used for storing the values of the standard CityGML attributes such as class, function, and usage etc. for building objects, while another spreadsheet can be employed to carry the energy-related attribute information like energy consumption and demand. A third spreadsheet may contain, for each building, a number of URL links of sensor-based services for querying real-time values e.g. the monitoring temperature and humidity. In map applications, once a city object has been clicked, all the linked online spreadsheets will be queried against the object id and the returned attribute information shall be displayed in an intuitive way that allows users to have a concise overview of all the information as well as to distinguish, which attributes belong to which domain or category.

To realize this, a conceptual framework has been developed in the course of the PhD research to define an abstract API allowing to build a map application for integrating different types of 3D visualization models and online spreadsheets. The structure of the API is shown as UML

diagram in Figure 92. Note that this UML diagram is meant to show the interrelationships between the diagram components. The attributes and methods of the individual classes and interfaces are omitted.

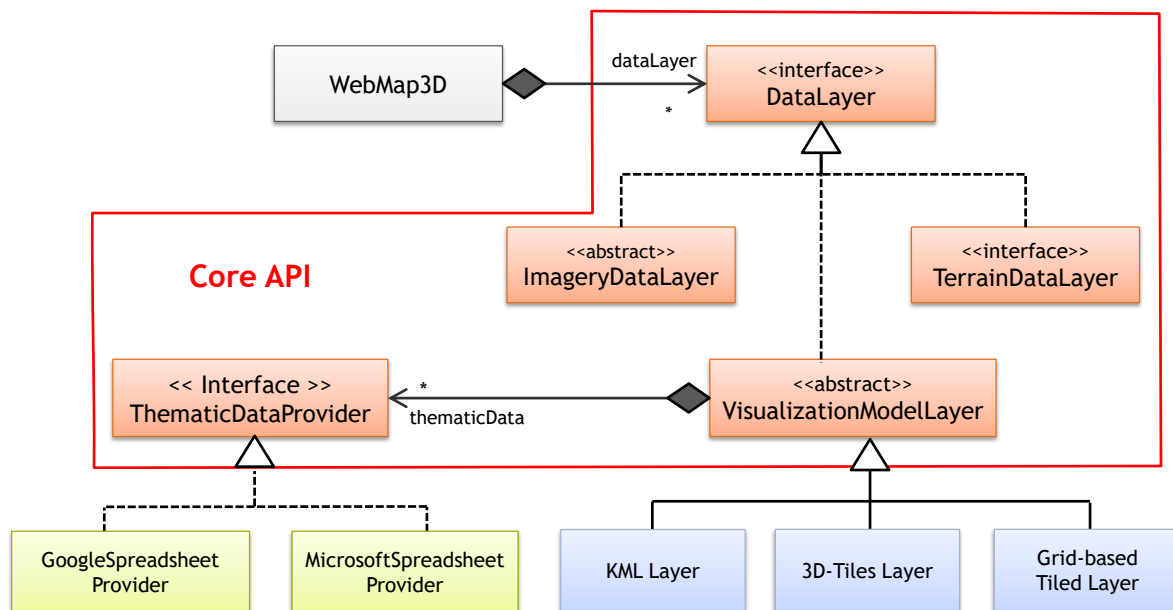


Figure 92: Conceptual API for coupling multiple online spreadsheets with a 3D visualization model

The class *WebMap3D* serves as a wrapper of a 3D map engine e.g. 3D virtual globe with the additional capability to hold different types of data layers e.g. imagery data layer, digital terrain data layer, and 3D visualization model layer via a common interface named *DataLayer*. This interface facilitates a map application to support the layer management functionalities like adding/removing, hiding/showing etc. of a selected data layer. In addition, the abstract class *VisualizationModelLayer*, which implements the *DataLayer* interface, can be extended to define the concrete classes that represent different types of 3D visualization model in the format like KML, gITF and 3D-Tiles etc. Each 3D visualization model class can comprise multiple thematic data providers by means of the *ThematicDataProvider* interface which exposes the function for constructing a HTTP request to fetch the thematic information of a city object based on its GMLID. Implemented classes of this interface can be e.g. Google spreadsheet or Fusion Table provided by the Google Drive Cloud Service and the Excel spreadsheet hosted via the Microsoft's OneDrive Cloud service etc. Further thematic data providers based on other Cloud or database services can also be defined by implementing the *ThematicDataProvider* interface. Based on this framework, a dynamically extendable 3D web application can be developed to support the rich exploration of 3D city models on top of a 3D virtual globe.

### 5.2.3 Implementation of a 3D Web Client

Based on the concepts introduced in the previous sections, a 3D web client has been developed. It is implemented on top of the WebGL-based Cesium Virtual Globe to make full use of the hardware acceleration and provides cross-platform functionalities like displaying 3D graphic contents on the web (cf. Chaturvedi et al. 2015, Yao et al. 2016). While developing the 3D web client, various extensions have been made to the Cesium Virtual Globe in order to facilitate users to view and explore 3D city models conveniently (cf. Figure

93). The major one among those extensions is that the high-performance visualization of large pre-styled 3D visualization models which can be formatted in KML, KMZ, or glTF formats and structured with different tiling schema e.g. 3D-Tiles or a simple grid-based structure. In addition, the rich interaction with the city models is also supported e.g. highlighting of 3D objects on mouseover and mouseclick, hiding and showing of the selected 3D objects as well as the exploration from different view perspectives using third-party mapping services like Microsoft Bing Maps with oblique view, Google Streetview, and a combined version using Dual Maps (cf. Figure 87). Moreover, the 3D web client implements the Cloud-based Google spreadsheet and fusion table as the online spreadsheet to store the thematic information coupling with the 3D visualization models and allows for querying the thematic data of every city object when it has been selected. In the past four years, this 3D web application has been successfully employed in many research projects and the core source code are managed using the Git version control system and can be freely obtained from the online Git-repository<sup>1</sup>. In the year 2015, this 3D web client won the first price in the 'Best Students Contribution' of the 'Web3D city modelling competition'<sup>2</sup> at the annual ACM SIGGRAPH Web3D Conference.

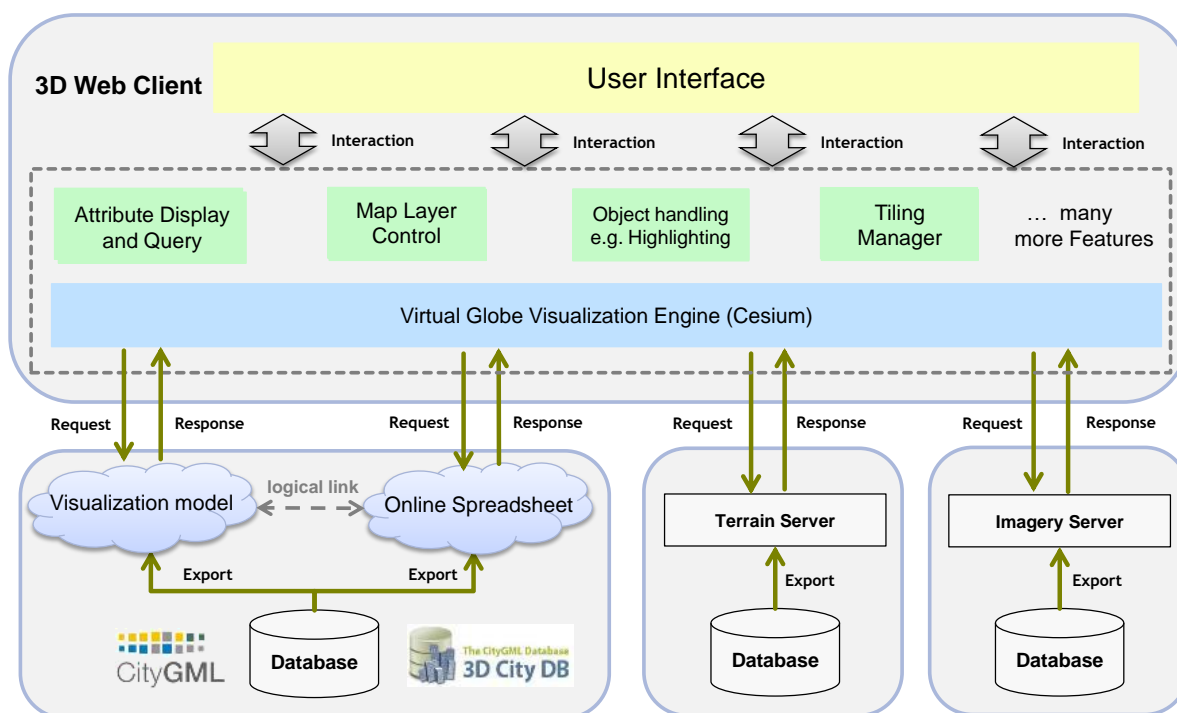


Figure 93: System architecture of the developed 3D web client

In most GIS applications, the term base layer (or basemap) is generally considered as a background layer on the map using, for example, satellite imagery and terrain model, to help people to quickly identify the locations and orientations from a certain camera perspective. In the 3D web client, a variety kinds of base layers can be visualized along with the 3D visualization models all of which can be conveniently managed via a common interface. Per default, Cesium comes with a number of selectable imagery layers provided by different mapping services, such as Bing Maps, OpenStreetMap and ESRI Maps etc. It is also possible to add additional imagery layer by using an OGC compliant Web Map Service (WMS) which

<sup>1</sup> <https://github.com/3dcitydb/3dcitydb-web-map>

<sup>2</sup> <https://www.ogc.org/blog/2274>

can easily be deployed using a number of open-source or commercial software tools like GeoServer, Deegree, and ArcGIS Server etc. Concerning the digital terrain model, Cesium Virtual Globe provides per default two available terrain layers. The first one is the so-called WGS84 Ellipsoid which approximates the Earth's surface using a smooth ellipsoid surface with a constant height value of 0. The other one is the so-called STK World Terrain using a worldwide 3D elevation data with an average grid resolution of 30 meters which is sufficient in many use cases. For specific application cases, high-resolution Digital Terrain Models might be required which can be stored and managed using spatial database, converted to a specific terrain format i.e. heightmap or quantized-mesh, and uploaded to a web server for the data access over Internet.

The user interface of the web client is created using the ExtJS JavaScript-based web framework which is freely provided by Sencha the under GPL3 open source license (cf. Sencha 2017). It provides a large number of GUI widget components such as grid table, popup window, Tab container, as well as tree-structured panel etc. which allow web developer to build an extensive web application with the support of rich user interactions. With the help of these outstanding features provided by ExtJS, the 3D web client has been designed very similar to most traditional GIS viewer applications to facilitate users to interact with working data and to manipulate the relevant operating functionalities for completing certain tasks conveniently. In addition, the main widgets of the web client were assembled component-wise and can therefore be easily customized to hide or disable some certain UI components for creating a lightweight version with concise user interface.

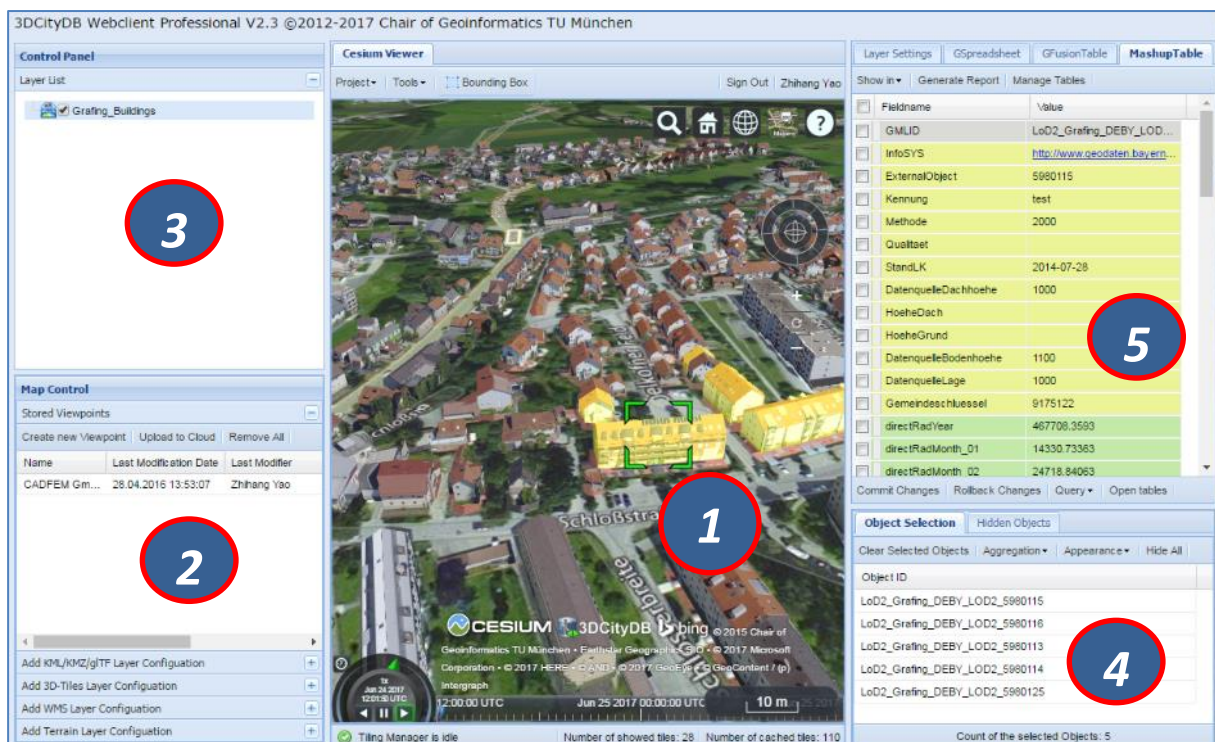


Figure 94: User interface and the relevant GUI components of the 3D web client

In the middle of the GUI (cf. Figure 94), a 3D viewer is embedded which is a standard Cesium widget that allows users to navigate the Earth view by panning, moving, tilting, and rotating the camera perspective using a mouse or touchscreen. In addition, the camera perspective can also be controlled by means of the Navigation Component which offers the



same navigation possibilities that can be achieved with mouse or touchscreen. It consists of a group of widgets, namely a Navigator widget for controlling the camera perspective, a North Arrows widget for orienting the Earth map towards the north, and a scale bar for estimating the distance between two points on the ground. It is also possible to switch the 3D view mode to 2D, which may result in a better rendering performance since all 3D objects will be automatically simplified and projected on the 2D map. Using the Map Control component [2], an arbitrary number of data layers can be added into the 3D web client and a name can be assigned to each added layer by the user. Once the 3D visualization models have been loaded and rendered on the 3D map, the loaded layer names will be listed in the Layer List panel [3] which allows users to interactively control the visibility of the loaded data layers or remove selected layers from the 3D web client.

Multiple 3D objects can be selected and highlighted on the map and the corresponding object IDs can be listed in the *Object Selection* panel [4]. When clicking on a single object, the thematic information will be queried from the linked online spreadsheets and the returned attribute information will displayed as key-value pairs in a tabular form in the table panel [5]. In order to distinguish which attributes are provided by which online spreadsheet, the table panel can be dynamically partitioned into several blocks each of which is painted with a distinguishable background color. Per default, the background of the attribute GMLID is coloured with grey color and the other blocks of the individual online spreadsheets can be flexibly configured by the users. To support this, a table manager component is available whose UI is shown in Figure 95.

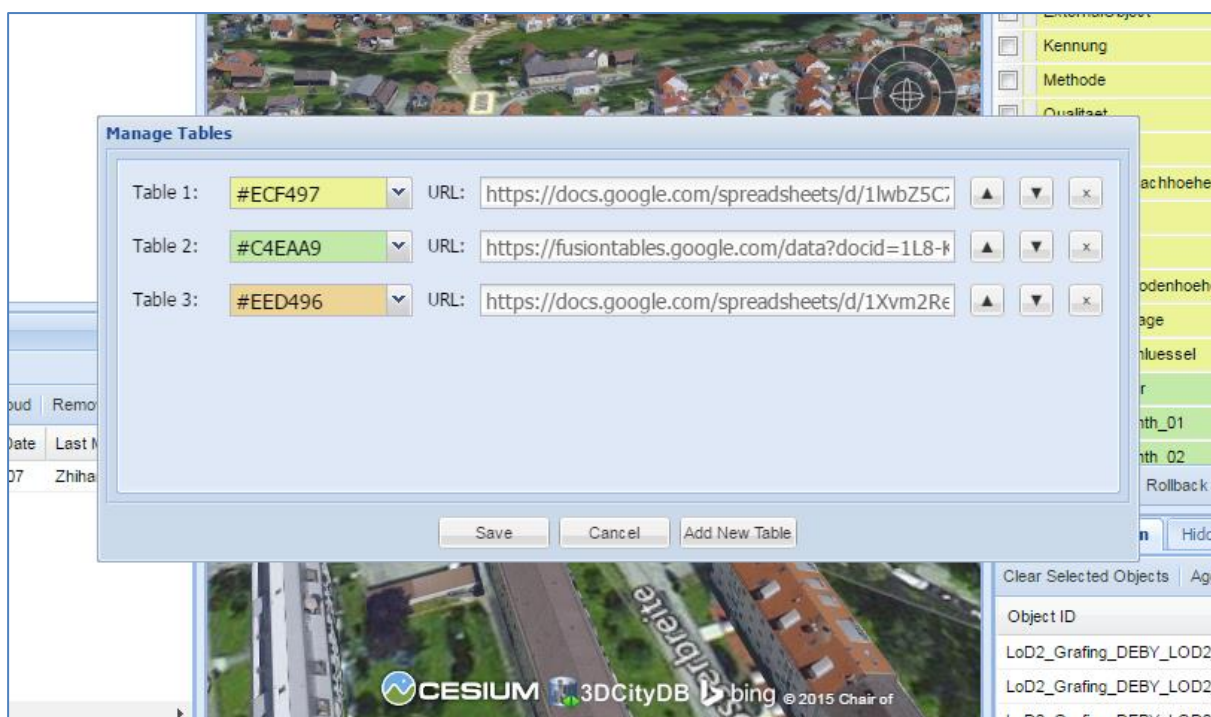


Figure 95: Table manager for dynamically adding multiple online spreadsheet to a data layer

Using this table manager, users are able to select their preferred background colors for the linked online spreadsheet and also to link additional online spreadsheets with the respective 3D visualization model. The display sequence of the respective panel blocks holding the attributes can be reallocated by clicking on the *Up* ( $\blacktriangle$ ), and *Down* ( $\blacktriangledown$ ) buttons of the

individual table. The *remove* (×) button allows users to remove an already existing item to disconnect an online spreadsheet from the respective 3D visualization model.

To ease the work with the 3D web client, the configuration information including the added data layers and their associated online spreadsheets can be stored in the Cloud by using an online spreadsheet called *configuration* spreadsheet. This spreadsheet has a fixed structure with a set of predefined columns and the relevant ones are shown in Figure 96. The first column with the name *LAYERID* holds the identifiers of the data layers and will be used by the 3D web client for indexing the data layers. Thus, these IDs must be unique in the scope of a configuration spreadsheet and can be assigned using the universally unique identifier (UUID). The layer names, which will be listed in the 3D web client, are stored in the column *NAME*. For each layer, the URL of the respective 3D visualization model can be found in the *URL* column, whereas the URLs of the linked online spreadsheets are stored in the column *Thematic Data URL* and separated with semicolons. With the help of a such configuration spreadsheet, an application workspace created by a user can be permanently stored, shared, cloned, and recovered via the Cloud. In addition, the URL of a configuration spreadsheet can be easily encoded as a key-value pair string to be attached to the 3D web client link, which can be stored as a browser bookmark or sent to project partners to facilitate the collaborative work (cf. Kolbe et al. 2003).

LAYERID	NAME	URL	THEMATIC DATA URL
UUID_1...	Roads	https://...	{https://...; https://...; ...}
UUID_2...	Buildings	https://...	{https://...; https://...; ...}
UUID_3...	Waters	https://...	{https://...; https://...; ...}
...	...	...	...

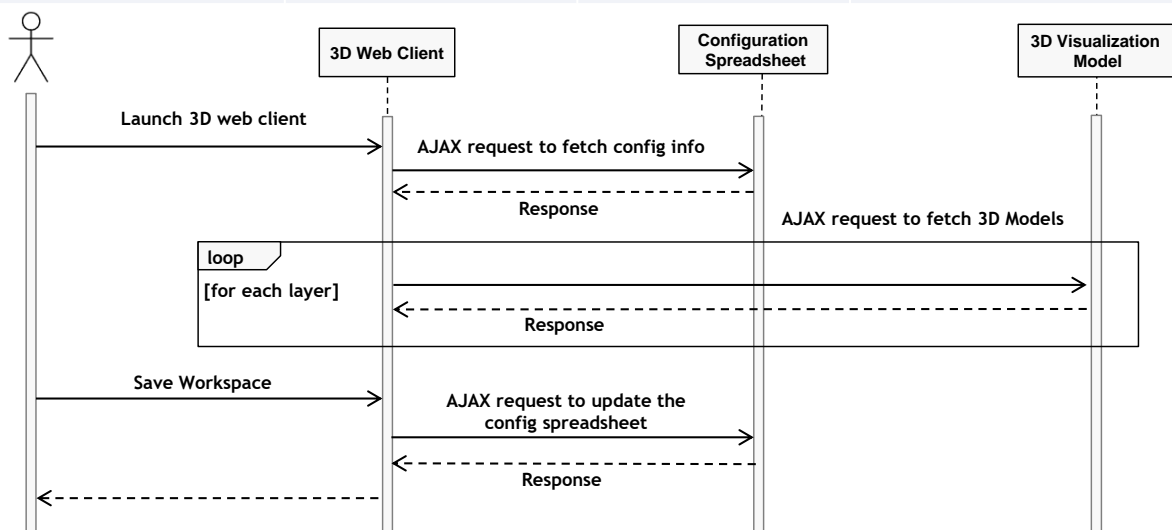


Figure 96: Idea of using a configuration spreadsheet for storing and loading linked 3D visualization models and online spreadsheets

The typical operation process of using the configuration spreadsheet in the 3D web client is summarized in the UML sequence diagram shown in Figure 96. When the 3D client is launched by the user, an AJAX request to the configuration spreadsheet is performed to return a response containing the layer information. Subsequently, the 3D web client iterates through

all the layer items and loads the respective 3D visualization models one by one. While operating the 3D web client, the user may add some new layers, remove certain existing layers, or enrich a layer by linking with additional online spreadsheets storing domain-specific information. After confirming the changes, the user can save the workspace to the Cloud by sending an AJAX update request to update the original configuration spreadsheet or create a new one.

The access to the configuration as well as the thematic data stored in the online spreadsheets can be easily controlled by taking fully advantage of the Cloud technology. Once an online spreadsheet has been created by a user, it can be shared with other users who may have either read-only or full access with write privilege to the spreadsheet. This allows multiple users to complete complex collaborative work by categorizing them into different user groups with different access rights. For example (cf. Figure 97), an application administrator creates a configuration file associated with two online spreadsheets storing the thematic information. Since the administrator is the owner of all the online spreadsheets, he can assign the write access to the user “data manager” who have full access to the thematic data spreadsheet but only read access to the configuration spreadsheet. The public users are only assigned with read access to the online spreadsheets allowing them to launch the web client and to query thematic information provided by the data manger. In this way, all the content information are properly secured among multiple users.

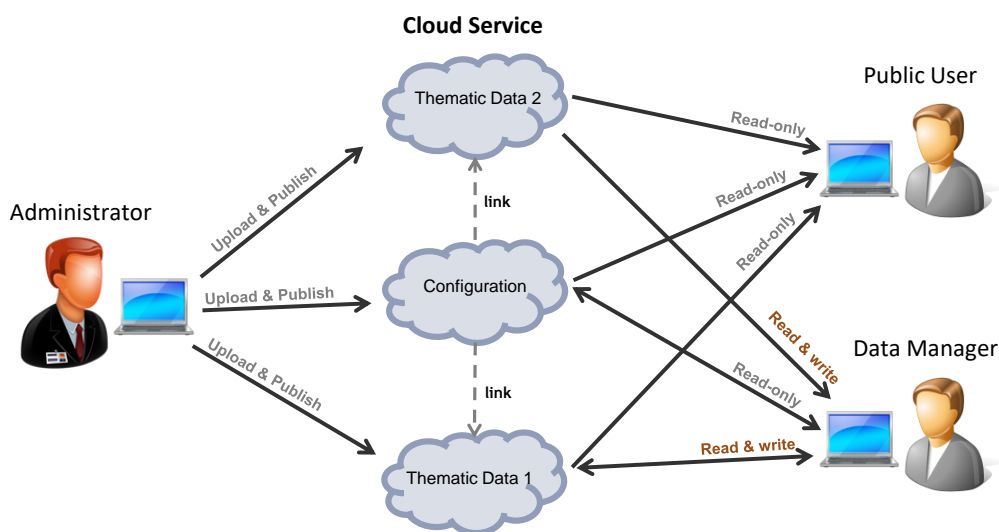


Figure 97: Collaborative work using the 3D web client based on Cloud-based online spreadsheets (cf. Herrerueta et al. 2012)

### 5.3 Example Applications

In this subsection three typical use case examples are introduced to show the usability and feasibility of the developed 3D web client in practical projects. Each of these examples has been intentionally chosen to demonstrate the individual key features of the 3D web client that have been successfully realized based on the research work and technical implementation. These results of these use cases were obtained in the course of different research projects which were carried out by the research team at the Chair of Geoinformatics at Technical University of Munich. As a software developer, the author was involved in all these projects

and mainly worked on enhancing the 3D web client according to the requirements of the project partners.

### 5.3.1 3D City Model of New York City

The project “3D City Model of New York City” is a student project which has been carried out in the context of three master theses of Barbara Burger, Berit Cantzler, and Christof Beil within the master's program Geodesy and Geoinformation at TUM (cf. Kolbe et al. 2015, Beil & Kolbe 2017). The key objective of these student projects is to create a homogenized and integrated semantic 3D city model of New York City (NYC) from the existing public 2D and 2.5D datasets provided in the NYC Open Data Portal. To reach this purpose, different spatial and semantic transformations and manipulations together with some photogrammetric analyses were investigated and performed using the ETL tool Feature Manipulation Engine (FME) of Safe Software. The resulting 3D city model is managed as a single CityGML file which comprises a variety of 3D feature types including all NYC buildings, land parcels, roads, parks, the digital terrain model, and water bodies and can be downloaded as Open Data from the project home page of the Chair of Geoinformatics<sup>3</sup>. In addition, the CityGML dataset has been imported into a 3DCityDB instance for the efficient data maintenance and also been converted to tiled KML/glTF models for the 3D visualization on the Virtual Globe like Google Earth and Cesium. To explore the resulting 3D city model conveniently, all the generated 3D visualization models have been deployed using the 3D web client and a screenshot of the online demo is shown in Figure 98.

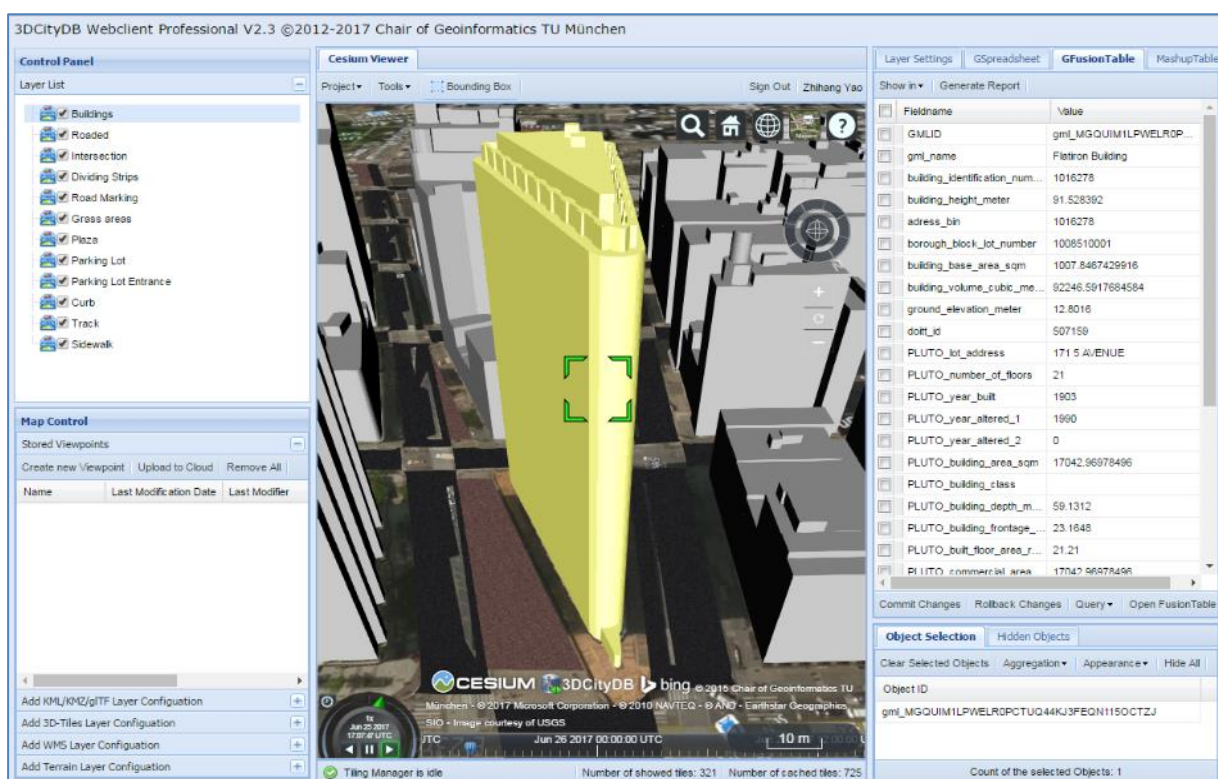


Figure 98: Example demo of visualizing 3D city model of New York City on the 3D web client

This 3D web client Demo contains all street space objects as well as all building objects of New York City in LoD2. More than one million building objects and more than 500'000 street

<sup>3</sup> <https://www.lrg.tum.de/en/gis/projects/3d-city-model-of-new-york-city/>



space objects were generated as tiled 3D visualization models separated into different layers according to the feature types. Moreover, each city object is enriched with a range of attribute information which have been uploaded and stored in the Cloud-based online spreadsheet ‘Google Fusion Table’ allowing for rich model exploration by clicking on the city object individually.

### 5.3.2 3D City Model of Berlin

Nowadays, a semantic 3D city model of the German capital Berlin is freely available to the public as open data in CityGML format. This 3D city model was created with the full coverage of the entire city area (890 km<sup>2</sup>) and comprises around 550,000 LoD2 building objects which are fully textured with the images photographed from the air (cf. Döllner et al. 2006). To date, this 3D city model is one of the largest textured CityGML models worldwide and can be obtained via a web-based service portal<sup>4</sup> through which users can select and download individual building models in a variety of 3D data formats in addition to CityGML, e.g. OBJ, KML, COLLADA, and 3D-Shapefile. Therefore, the 3D city model provides a good foundation for the research and development of Smart Cities. For example, in the context of the research project “Energy Atlas Berlin”, this 3D city model has been utilized as an important information basis for developing the tools for holistic calculation of the energy consumptions, demands and saving potentials at an urban scale. The results of the calculation and simulations can be enriched to the 3D city models and shall be able to be explored on a 3D map. This has been realized using the developed 3D web client and a screenshot of a dedicated online demo is shown in Figure 99.



Figure 99: Example of visualizing 3D city model of Berlin on the 3D web client

In this example, each building object has been generated with three different display forms starting from the coarse “Extruded” geometry, via the untextured geometry up to the most

<sup>4</sup> <https://www.businesslocationcenter.de/downloadportal/>

detailed textured geometry. Depending on the distance between the camera and the individual building object, one of the three geometry representations can be automatically chosen by the 3D web client for the display and can also be dynamically switched during the runtime according to the Level of Detail mechanism: The buildings being far away from the camera are simply rendered as extruded or untextured geometries, while the textured buildings with higher details will be rendered when viewing the building objects from a short distance. In this way, the efficient 3D visualization of the 3D city models with higher level of details ( $LOD \geq 2$ ) is possible.

### 5.3.3 3D City Model of Vorarlberg

In the context of the VoDLM3D (*Semantisches 3D-Landschaftsmodell Vorarlberg*) project, the Vorarlberg State Government plans to establish and provide a semantic 3D landscape model which shall contain comprehensive 3D spatial and thematic information on roads, water bodies, vegetation coverage, as well as topographic objects like buildings, bridges, tunnels, ski slopes, high-voltage lines and towers (cf. Marx et al. 2017). All these data shall be able to be administrated by the Vorarlberg government and can also be publicly accessible to the citizens from Vorarlberg based on a 3D web solution. For this purpose, the existing 2D/3D geospatial data of Vorarlberg have been first transformed to a CityGML data model using the software tool FME and then converted to the respective 3D visualization models with different data layers according to the topographic feature types. A subset of the project results is represented as a 3D web client demo which is accessible via the Github<sup>5</sup>. This demo (cf. Figure 100) shows around 9800 attributed LoD2 buildings along with an imagery base map according the OGC Web Map Service (WMS) as well as a high-resolution (0.5 meter) digital terrain model resulted from converting a GeoTIFF-formatted DTM to the Cesium-compliant *heightmap* format.



Figure 100: Example of visualizing 3D landscape model of Vorarlberg on the 3D web client

<sup>5</sup> <https://github.com/3dcitydb/3dcitydb-web-map#demos>

## Chapter 6 Utilization of Domain Extendable 3D City Models

In the previous chapters, the relevant concepts and approaches regarding the efficient management, visualization, and exploration of large and extendable 3D city models are elaborated to show the possibilities of utilizing 3D city models in real-world application cases. In the context of smart city projects, the growing complexity of the application scenarios strongly requires a complete and integrative platform which should be capable of performing the analyses, simulations, and modifications on the complex-structured 3D city models. Certain parts of such procedures should be operable not only through a time-costly off-line processing step but also via a simple user interface during the runtime of the 3D visualization and exploration. This will allow both GIS experts and non-experts to easily access and manipulate the data models as well as to rapidly capture the simulation and analysis results to help them in completing the decision-making process (cf. Shen & Kawakami 2010). Moreover, since all the data models are organized in a central database based on a common model definition, the events of data changes and simulation results can be automatically propagated and spread to the other application sectors to accomplish the collaborative tasks of urban management (cf. Hofman et al. 2011).

Nowadays, an increasing number of analysis and simulation tools have been developed for the application domains like urban planning and building energy simulation (cf. Sousa et al. 2012). Many of these calculations can produce rich output information which can be automatically derived from the semantic 3D city models and will be analyzed by decision-makers such as city planners and architects etc. to make decisions by comparing the different results of the calculations upon the different input parameter values. However, these users usually have very limited knowledge about the GIS technologies or the complex data structures behind the calculation logics and their main task is to analyze some particular indicator values which directly result from simulations. Thus, they do not need to dig deeper into the complex methods to handle calculation procedures from scratch, but just need to focus on the interrelationships between different input parameter and indicator values in order to simulate different situations to find out the optimal construction measures and plans for improving the future energy supply.

For this reason, such decision support systems for dealing with analysis and simulation requires that, both complex data models and simulation engines must be organized with an integrative framework to ensure 3D city models are accessible by normal users. On the one hand, this system should provide a simple user interface for the normal users allowing them to view, explore, edit, as well as analyze the 3D city models without the need to access the original 3D city models directly. On the other hand, the modified data as well as the simulation results can be written back to the original 3D city models which can be used as the input for other simulations of different application domains. To realize this, the design of a complete and scalable system platform is of great importance to assist a wide group of users to easily access and handle the 3D city models during the planning, design, and development process collaboratively.

In this chapter, emphasis is placed on the introduction of a novel approach of a new system architecture concerning the requirements mentioned above. Additionally, technical implementation of the system architecture is elaborated, and two typical application examples

based on the system architecture are given for proving the feasibility of the introduced approach.

## 6.1 Conceptual Considerations

### 6.1.1 Problems of the traditional System Solutions

Nowadays, a number of open source and commercial application systems were brought to the market, which support working with semantic 3D city models for a range of application scenarios in many cities (cf. Hildebrandt 2014). Most of these applications are constructed based on a so-called three-tier architecture (cf. Figure 101) which utilizes a data tier at the lowest level for dealing with the storage of 3D city models in a spatially-enhanced database management system. The communication between client application of the presentation tier and the data tier is realized through an application tier also called middle tier which contains the main business logic and various functionalities like session management for controlling and performing the complex tasks including the query, analysis, as well as CRUD (create, read, update, and delete) operations on the stored data contents. To minimize the complexity of implementing such Three-tier architecture, a number of existing web application frameworks such as Spring and Django etc. are available for the development and deployment of the entire system through a set of standard templates and libraries. Moreover, the application tier can be implemented as a web service according to the standard web interfaces i.e. Web Feature Service (WFS) and Web Processing Service (WPS) which allow to access the application tier via a URL in a way of sending a standardized HTTP request to the business logic for performing the CRUD operations and analysis procedures and receiving back a response containing the respective results.

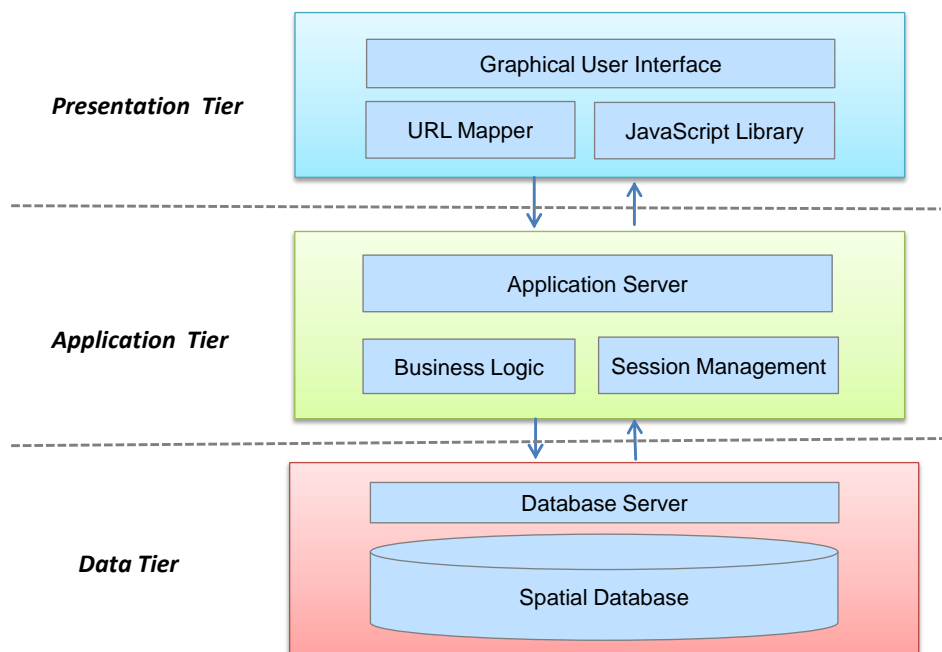


Figure 101: Traditional Tree-tier architecture of a 3D Web GIS application (cf. Westra 2010)

This architecture approach works well for most application cases but has certain limitations regarding two important aspects. First, most application users who are not authorized to have write-access to the application tier and data tier are restricted from modifying the data



contents stored in the central database. This is because, the stored 3D city models are the important information basis for performing the application logic and, hence, must be kept consistent and can only be altered by advanced users having administration privilege. However, in some application cases, the normal users may have to make some changes and updates to the 3D city models which shall be temporarily stored in a form like a “pull request” which can be checked and validated by the application administrators and merged back to the central database in a later stage. Second, depending on the application scenarios, users may want to add some new data and functionalities to the application system. However, the application tier is usually implemented as a behind-the-scenes module which is difficult to be modified or redeployed by the users due to the high system complexity in terms of the hardware and software requirements. Considering the two above-mentioned limitations, a new architecture approach is needed to ease the utilization of 3D city models for broader user groups.

### 6.1.2 A new Multi-Level System Architecture

In the context of this dissertation thesis, a new multi-level system has been developed which provides a novel conceptual solution for constructing a scalable 3D GIS environment and has also been successfully implemented using the existing software tools and 3D city model standard. The entire structure of this system can be subdivided into three logical tiers namely the ‘information backbone’, the ‘application level’, and the ‘end user level’ shown in Figure 102 (cf. Yao et al. 2014).

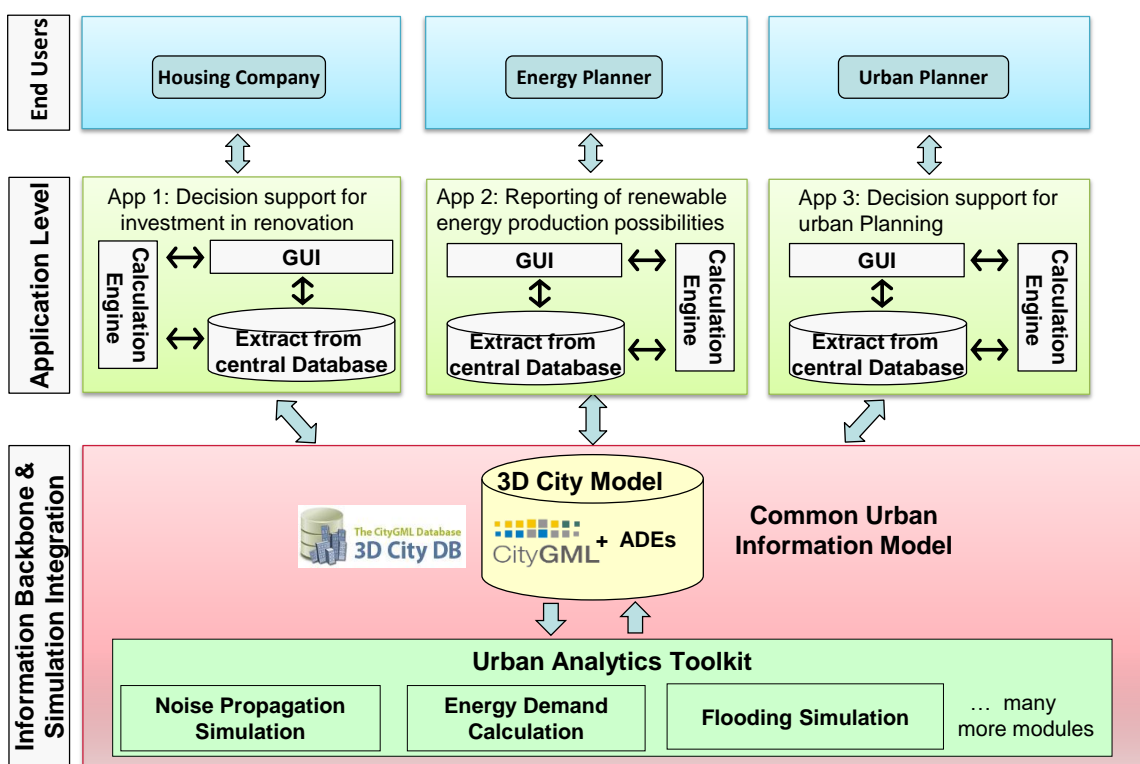


Figure 102: Three-tier system architecture (cf. Yao et al. 2014)

At the lowermost level (Information backbone), the semantic 3D city model is created which contains the city level information from different sources such as CityGML, Industry Foundation Classes (IFC), ESRI shapefiles, CAD files and others. The city information from

different sources can be integrated into the common data model and exchange format according to the CityGML standard along with its ADE mechanisms. This information model shall be imported into a spatial database allowing for efficient storage and management of the large 3D city models which can be efficiently accessed by different application domains. In addition, an urban analytics toolkit can also be invoked, which is a bundle of complex functional modules for performing the time expensive computations i.e. the derivation of key indicator values, the attribute aggregation calculations, and the morphometric analysis etc. which are essential in many different application fields like urban planning, environmental and training simulations, disaster management, and energy assessment. Moreover, since all these toolkits share the same data basis, multiple modules can be used together to complete more complex tasks.

The *Application Level* acting as a “bridge” between the *End Users* level and the *Information Backbone* is introduced which shall provide easy-to-use applications to the specific end-users from different domains. Since most users from the *End Users* level usually have very little GIS expertise and have nearly no knowledge of the complex 3D city models, they need only intuitive tools that allow them to access the relevant part of the semantic 3D city models and to perform some certain ad-hoc data modifications, simulations and calculations. Thus, all applications at this *Application Level* must be designed in such a way that they follow the principle of the "App concept", which has been established in recent years in the context of smartphones and tablet PCs. This principle requires that for specific users or user groups, each application should have a very limited range of functions in order to be relatively easy to learn and intuitive to use. In addition, each application shall also be portable and can be easily modified, recovered, or even cloned to a new one.

In order to create an application in compliance with the “App concept”, a number of simplification procedures are needed to minimize the complexity of the application. First, it is important to make clear about which functions provided by the urban analytics toolkit are required by the end users and what information of the original 3D city model are relevant for the application, because the required data and analysis functions may vary depending on the application and target users of different application domains. In the subsequent step, the relevant data contents shall be extracted from the central database according to different filter criteria such as spatial bounding box, feature classes, and level of details etc. The corresponding spatial content can be exported as tiled 3D visualization models and pre-cached at the server side, whereas the thematic information can be mapped onto a simple table structure allowing for easy data access and to realize the logical link with the generated 3D visualization models. In addition, a calculation engine serving as a light-weight version of the urban analytics toolkit shall be embedded into the application for performing the required application functions. Moreover, the application level shall provide a user-friendly graphical interface that helps users to explore the exported spatial and thematic information as well as to operate the calculation engine to accomplish the application-specific simulation and analysis tasks.

During the runtime of the application, the updated data content as well as the results of the calculations and simulations are temporarily stored at the application level. Therefore, it is very important to be able to import these up-to-date data back into the central database. The application administrator or project manager must be able to determine what information

contents have been changed, added, or deleted compared to the current database. After confirming all the changes, a bulk process for updating the 3D city models in the central database can be performed and the updated 3D city models can be used by further applications and users of different domains.

## 6.2 System Implementation

The conceptual idea of the newly introduced multi-level system can be fully implemented for the practical application cases by utilizing the existing GIS, ETL, and database-based software tools including the 3DCityDB software toolkit along with the 3D web client introduced in the previous chapters. The entire system implementation consists of two parts according to the system levels: one is the implementation of the information backbone and the other one is the implementation of the application level. In the following two subsections, the relevant aspects about the technical implementation of the both system levels are illustrated respectively.

### 6.2.1 Implementation of the Information Backbone

A general way, among other things, of using the CityGML standard and the existing software tools for creating the information backbone as well as the urban analytics toolkits is shown in Figure 103. Since most of the applied software tools, especially the 3DCityDB software package, have already been explained in the previous chapters, only the relevant workflow of using the outlined tools and the comparison of different approaches are illustrated here.

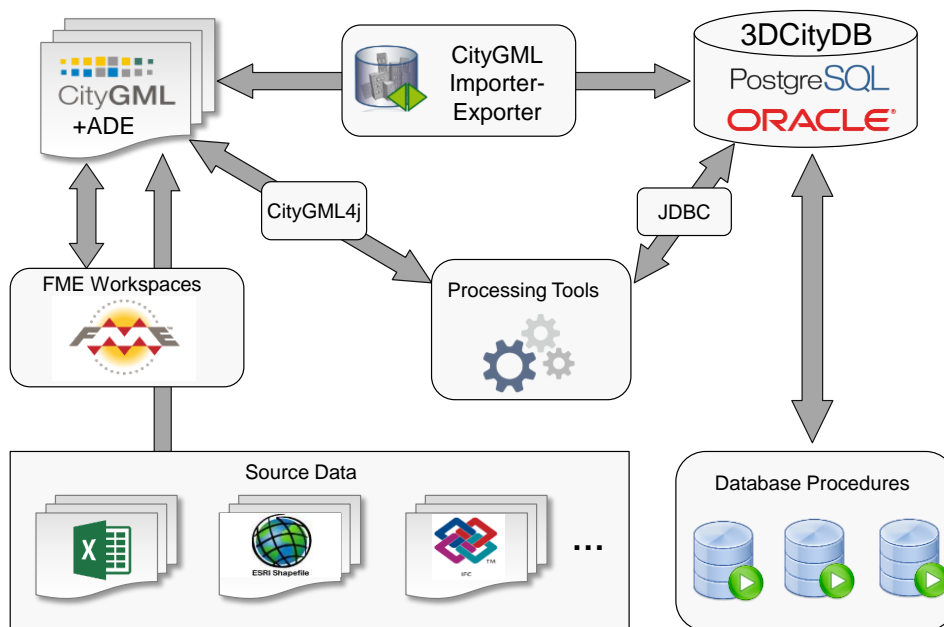


Figure 103: Workflow of setting up the information backbone and urban analytics toolkit

To integrate the diverse spatial and non-spatial data with different formats into a common data model in CityGML, the software tool Feature Manipulation Engine (FME) can be used which allows the transformation of different data sources such as Industry Foundation Classes (IFC), ESRI shapefiles, Spreadsheet files and others into CityGML and vice-versa. For the creation of CityGML instance documents with ADE data, FME provides a transformer tool

called 'XMLTemplater' which can parse and interpret the ADE's XML schema definition file and use it as a template for creating instance documents, which are ensured to be valid against the parsed ADE schema. In addition, FME offers a drag-and-drop user interface empower users to design and develop their own analytics tools by assembling a set of transformer functions provided by FME to realize complex data processing workflows. The calculation results can be embedded into the CityGML model using either CityGML's Generics module or the ADE model, and exported as a single XML document for the data transmission and exchange.

For the persistent data storage and maintenance, the CityGML file can be imported into a 3DCityDB database instance using the CityGML Import/Export tool. At the database level, it is also possible to design a variety of powerful urban analytics toolkits by means of the database procedures written using the script language such as PL/SQL for Oracle and in PL/pgSQL for PostgreSQL/PostGIS. One of the advantages of using the database procedures is that, developers can have a high flexibility in designing the analysis functions programmatically and also can efficiently query the required data by just accessing the corresponding database tables with the help of the indexing capabilities of the database to achieve a sophisticated processing performance. In addition, the database procedures can also be rapidly developed by utilizing the existing spatial function packages provided by the spatial database and allows to develop complex spatial analysis and operation toolkit with concise code structure.

The implementation of an urban analytics toolkit can also be done using high-level programming languages like Java, Python, C#, and C++ etc. Compared to the database procedures, such processing tools are able to provide a graphical user interface allowing users to interact with the application program in an intuitive way. Besides, the data access to the CityGML data contents is also very simple and flexible. For example, the data access to the 3DCityDB database can be realized by means of the Java Database Connectivity (JDBC) library which is a standard part of the Java Development Kit (JDK) and provides a common interface and methods to query and update data in a database. Moreover, the Java program can also directly call the database procedures installed the 3DCityDB and receive the processing results via the JDBC. Regarding the access to CityGML instance document, the Java-based library `citygml4j` is a powerful means for reading, validating, processing, and writing CityGML documents. This way, an extensive urban analytics toolkit based on the 3D city models can be developed and built as a single runnable program that can be launched on any computer installed with a Java virtual machine or serves as a function library referenced in other toolkits.

### **6.2.2 Implementation of the Application Level**

The implementation of the Application Level can be facilitated by making use of the Cloud technology which fits very well to the technical requirements of setting up an application for specific user groups. For the graphical user interface, the 3D web client introduced in the previous chapter has been extended with the full functionalities for interacting with the Cloud-based online spreadsheets which can not only serve as an online repository for storing the thematic information of 3D city models but also provide a sophisticated calculation engine



for performing many kinds of data processing and calculation tasks such as data modification, query, statistics, and ad-hoc calculations.

For instance (cf. Figure 104), when using Google Spreadsheets, the city objects displayed on the 3D web client can be queried on the basis of their attributes stored in the online spreadsheet, which supports a set of query functions allowing to retrieve the desired data records using a query expression that has a very similar syntax as the Structured Query Language (SQL). The query statements can be encoded with the spreadsheet URL and then sent to the spreadsheet server via a simple HTTP call. In addition, since only authenticated users have access to the target online spreadsheet, the request must additionally carry a so-called *token* code which will be interpreted by the Cloud server to determine, if the request is sent from an authorized user. Once the Cloud server has completed the query processing, a response containing the query results will be formatted in a JSON structure and can be easily parsed at the client side using JavaScript engine.

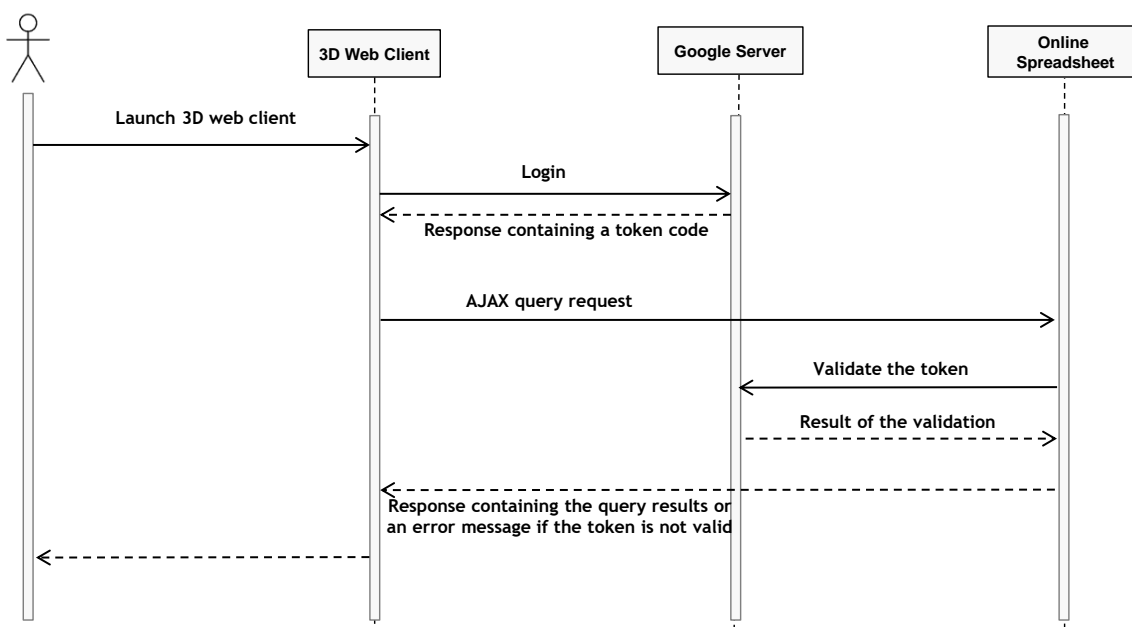


Figure 104: Sequence diagram of the process flow when performing a query on an online spreadsheet

In the online spreadsheet, the geographic coordinates of each city object's centroid can be stored in two separate columns which represent the longitude and latitude attributes respectively (cf. Figure 105). By applying a simple query against the two columns, it is very easy to find out those city objects whose center points are contained within a 2D bounding box which can be defined by simply selecting two diagonal points on the map. The query statement may look like the followings:

```
SELECT * from [tableName] WHERE Longitude > minX AND
Longitude < maxX AND Latitude > minY AND Latitude < maxY;
```

Where the value pairs [minX, minY] and [maxX, maxY] denote the coordinates of the diagonal points of the given bounding box in WGS84 Geographic coordinate reference system. In case that a polygon area is given, its maximum bounding box can be first calculated to be used for performing a pre-filter and the returned city objects will be further checked at the client side using a *ray-crossing* algorithm to determine which city objects are located within the polygon. This function is especially useful when, for example, developing a

web portal which should offer users the possibility to define a polygon and download all the spatially related 3D city objects.

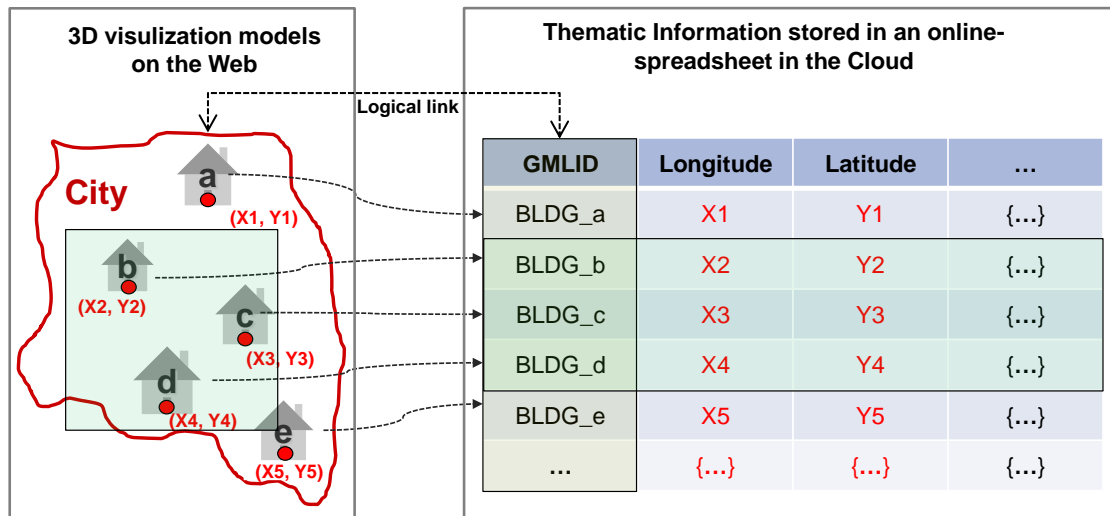


Figure 105: Conceptual idea of using online spreadsheet to perform a simple spatial query

In addition to the query function, the data contents in the online spreadsheet can be updated via a simple authorized HTTP request. Once the value in a column has been updated, these changes can be automatically propagated to another column which is defined as the result of a formula based on the values from other columns. A simple example is shown in Figure 106 to illustrate the basic concept.

**Thematic Information stored in an online-spreadsheet in the Cloud**

GMLID	Attr1	Attr2	Attr3
BLDG_a	400	4	100
BLDG_b	500	5	100
BLDG_c	600	6	100
BLDG_d	700	7	100
BLDG_e	800	8	100
...	...	...	...


  
**Formula: attr1 = f(Attr2, Attr3) = Attr2 × Attr3**

Figure 106: Example of creating a simple calculation engine using an online spreadsheet

In this example, three columns *Attr1*, *Attr2*, *Attr3* are defined in the online spreadsheet representing three different numeric attributes of a 3D city model. A simple formula has been defined on the column *Attr1* whose value at each row must be automatically resulted from multiplying the values of the *Attr2*, *Attr3* columns and can therefore not be freely changed by the users. Once a value in the column *Attr2* or *Attr3* has been changed, then the value of the

corresponding cell in the column *Attr1* will be automatically updated. Similar to most desktop spreadsheet calculation programs like Microsoft Excel, the Cloud-based online spreadsheet i.e. Google Spreadsheet provides a rich set of functions for designing a formula which can be used for developing a variety of calculation engines for specific applications.

However, using online spreadsheet as a calculation engine requires that the spreadsheet must be editable, and the users must be authorized to have write-access to it. This would be very risky regarding the data security and consistency, since this online spreadsheet may also contain other columns for storing the important attribute values which can only be edited by the application manager. To overcome this issue, a specific strategy using two interrelated spreadsheets has been developed which is illustrated using a simple example (cf. Figure 107). As shown in the figure, all attributes, which must be changeable by the end user to run the calculation, are extracted and stored in a separate spreadsheet (Spreadsheet 1). The end user shall have full access to this spreadsheet and can arbitrarily change the contained values. The second spreadsheet (Spreadsheet 2) can be seen as the main spreadsheet where the calculation formula is defined and in which the unmodifiable attribute data are stored. Thus, the end user can only have read-only access to this spreadsheet. The columns *Attr2* and *Attr3* are linked with their counterparts in the first spreadsheet and their values can be automatically updated once the referenced column values in the first spreadsheet have been changed. To establish such reference relationship, the owner of the second spreadsheet must be granted to have read access to the first spreadsheet for being able to pull the data from it. In this way, the calculation engine can be run by changing the input values in the first spreadsheet and the calculation results along with the other attribute information can also be well-protected as well as accessible to the public users.

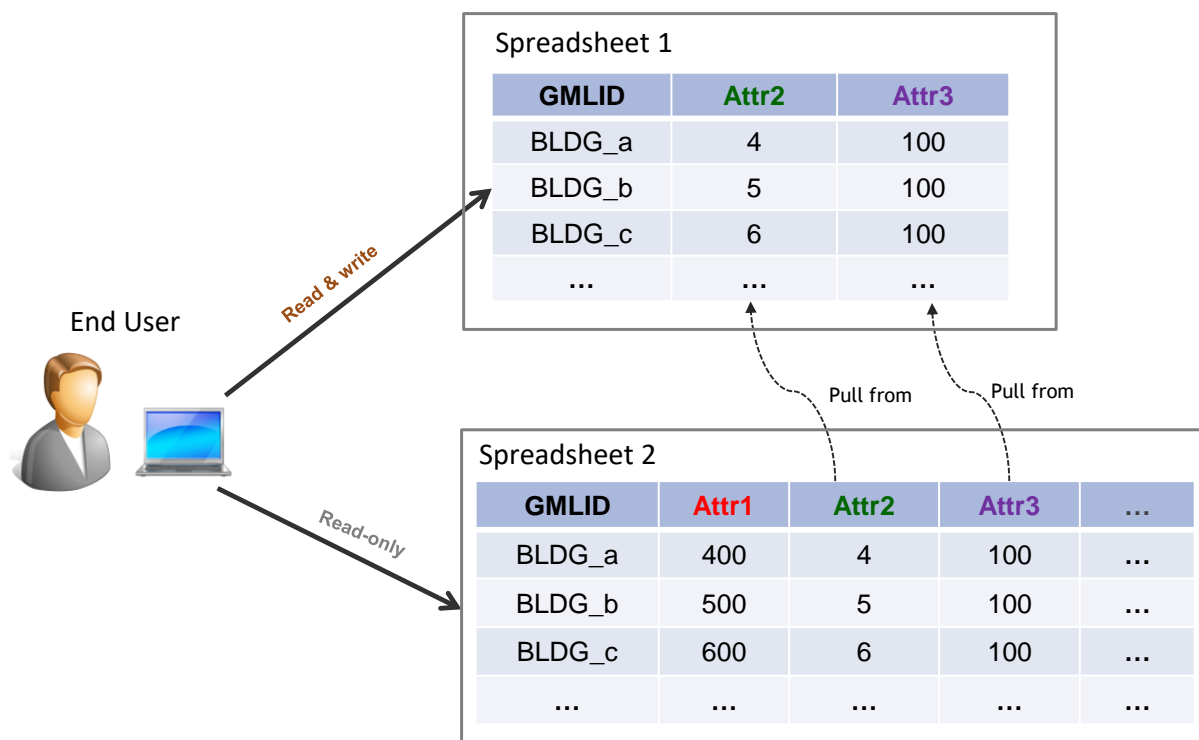


Figure 107: Example of using two separate spreadsheets to perform calculations and simulations with secured data

## 6.3 Example Applications

The new multi-level system has been implemented, used, and evaluated in several commercial and scientific projects within various domains e.g. solar irradiation analysis, energy demand estimations, and identifying retrofitting and refurbishment potentials for buildings. In this section, two corresponding use cases along with the 3D web client applications are presented.

### 6.3.1 Use Case 1: Solar Potential Simulation for the district LBBD

In the context of this commercial project, a 3D web application was required which should be able to provide services in support of the ESCO (*Energy Service Company*) strategy that specifically identifies the solar potential within the district London Borough of Barking and Dagenham (LBBD). In this context, a CityGML-based 3D city model of the buildings in the LBBD district was created to serve as a common information hub for carrying the solar potential information. These information shall be efficiently managed and visualized in order to allow users to explore and analyse the solar potential value of the individual buildings as well as their other relevant attribute information e.g. size, shape, height, ownership, and usage etc. The process for developing such information model consists of two major steps: The first step is the creation of 3D building models according to the CityGML standard. The second is to calculate of the solar potential value for each building in the district LBBD and to enrich the 3D building models with the calculation results.

In the first step, the 3D building reconstruction was done by means of an automatic process using the 2D footprint datasets of the building object along with the high-resolution Digital Surface Model (DSM) and Digital Terrain Model (DTM). While the DTM describes the actual shape of the Earth surface, the DSM represents the real height of the surface situations with respect to the Earth surface as well as the 3D urban and landscape objects on it. Higher resolution ( $\leq 0.5\text{m}/\text{grid}$ ) of the two models enables the fine-grained reconstruction of the roof surfaces of 3D building objects and can hence generate LOD2 building models according to the LOD definition in CityGML standard (cf. Haala & Kada 2010). This reconstruction process is well supported by the commercial software tool developed by the company virtualcitySYSTEMS.

The calculation of the solar potential values was done by using the Solar Potential Analysis Tool (cf. Chaturvedi et al. 2017) developed by the team of the Chair of Geoinformatics at TUM. It is a Java-based simulation tool allowing for estimating solar energy production for roofs and facades of buildings through a multi-threaded process. Based on the CityGML 3D building model together with the calculated sun position and an approximated sky dome, the solar power from direct, diffuse, and global solar irradiation for each month of the year have been estimated for individual building with respect to the shadowing effects of its surrounding topographic objects. Moreover, the Sky View Factor (SVF) which is a value ranging between 0 and 1 for representing the fraction of visible sky on a hemisphere has also been calculated for all surfaces of each building.

The created CityGML dataset containing the calculation results of the solar potential analysis were first imported into a 3DCityDB instance. Next, the thematic information were exported and uploaded to a Cloud-based online spreadsheet using the Google Fusion Tables. A

corresponding 3D visualization model has been generated with an LOD2 geometry representation and can be visualized in the 3D web client as shown Figure 108.

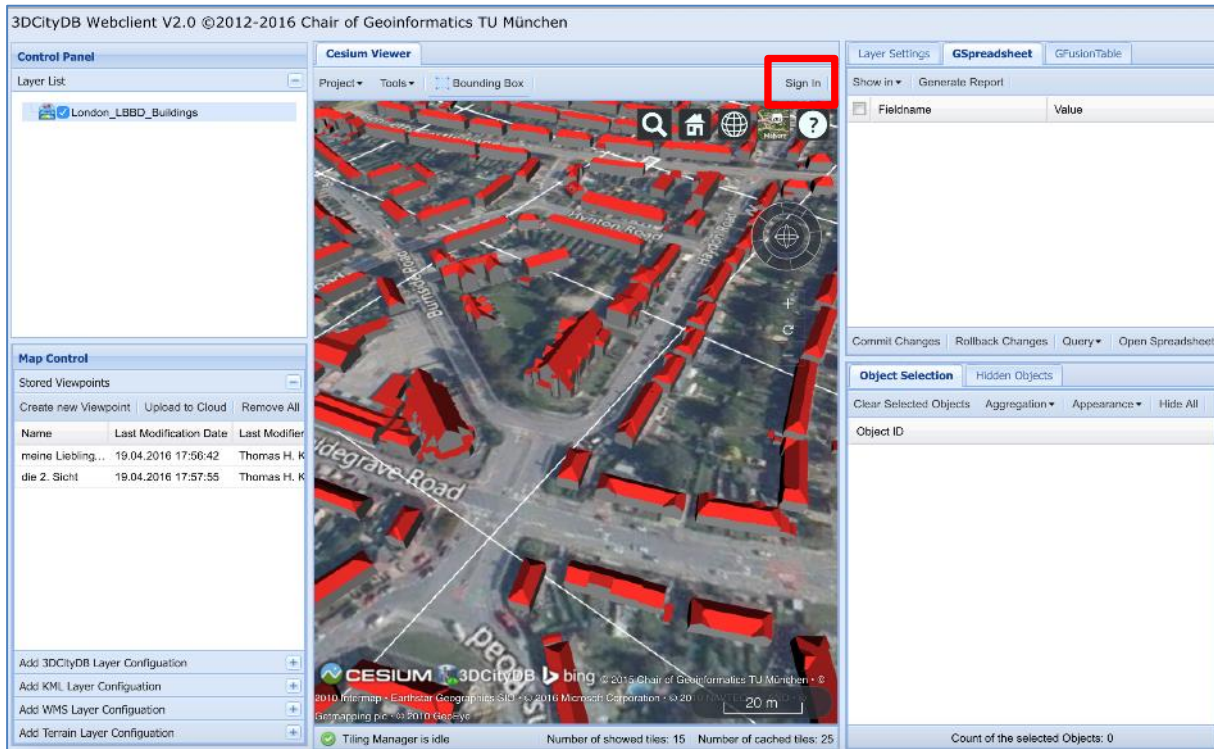


Figure 108: User interface after loading the LBBB 3D building models into the 3D web client

To access the content information stored in the Google Fusion Table, authorized users must sign in with their Google account by using a prompted login dialog window (cf. Figure 109).

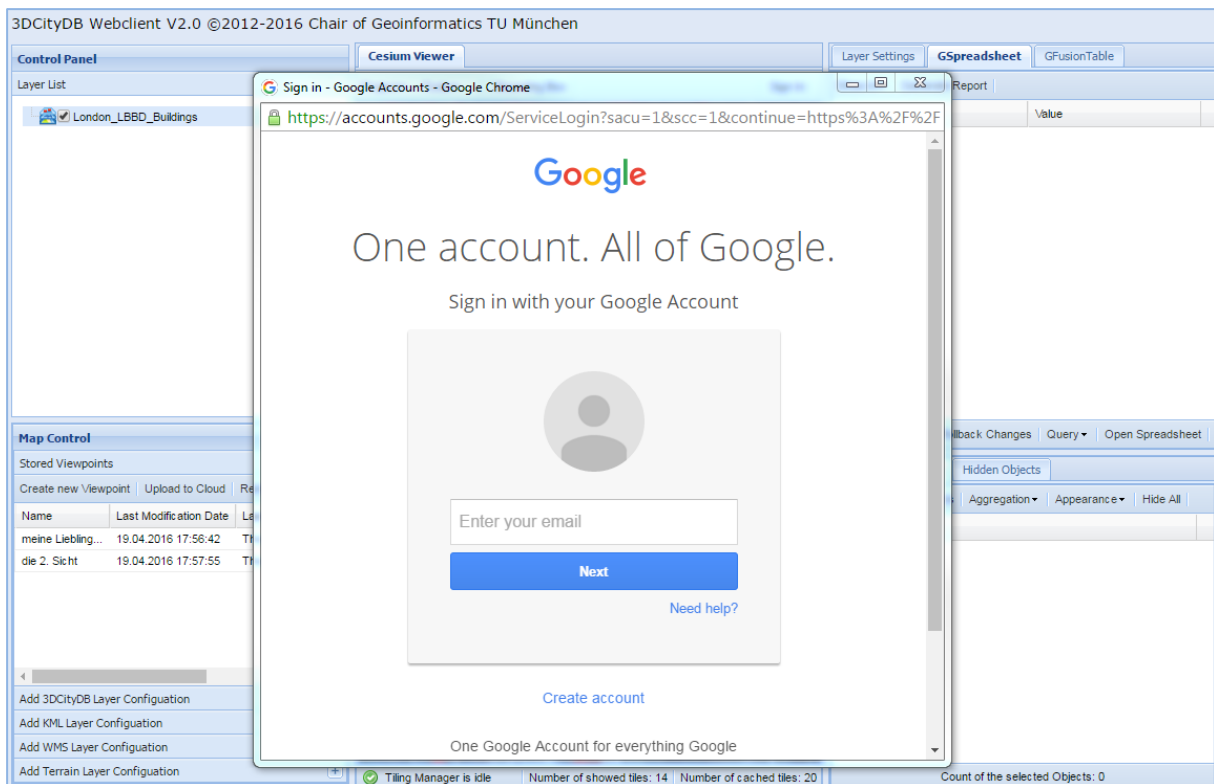


Figure 109: User-dialog for signing up using a valid Google Account



Spatial queries can be performed by selecting a bounding box and the query results are shown in Figure 110.

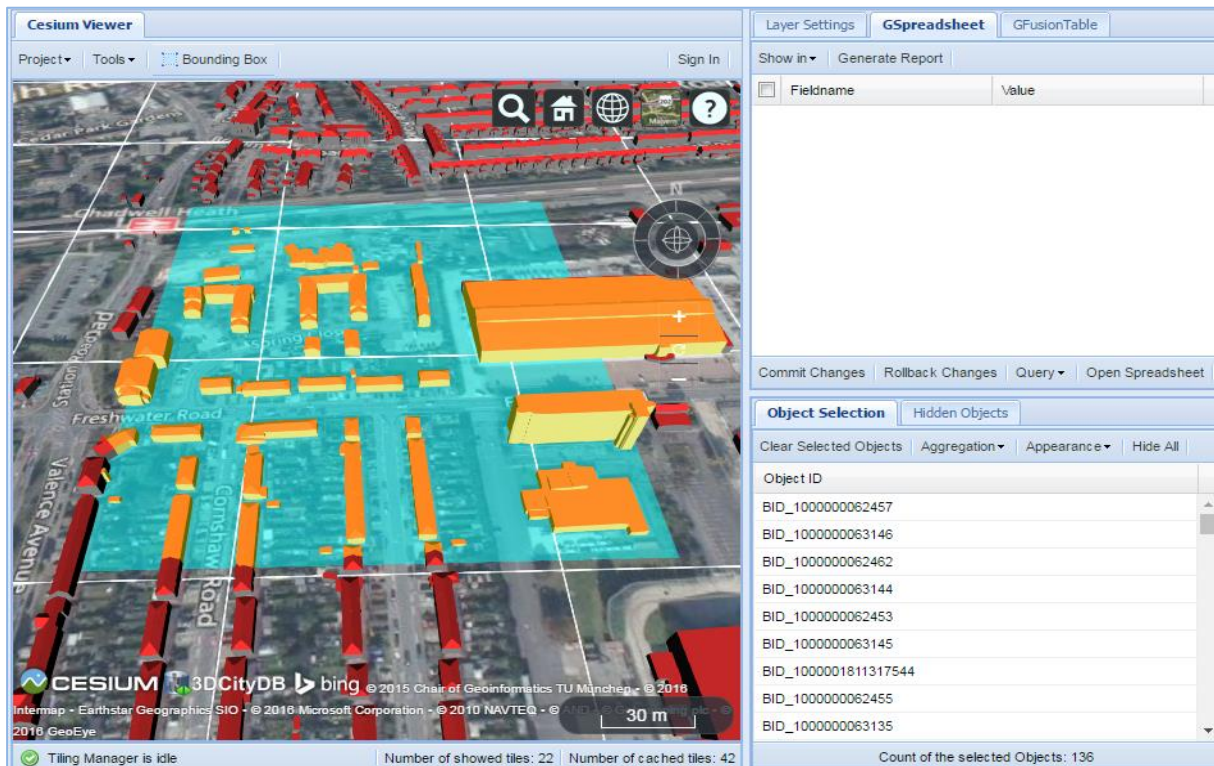


Figure 110: Example of performing a spatial query using a user-defined bounding box

As an example of the query operation, all buildings along “**HAYDON ROAD**” are queried and a total number of 83 building objects are returned and highlighted in the 3D web client (cf. Figure 111).

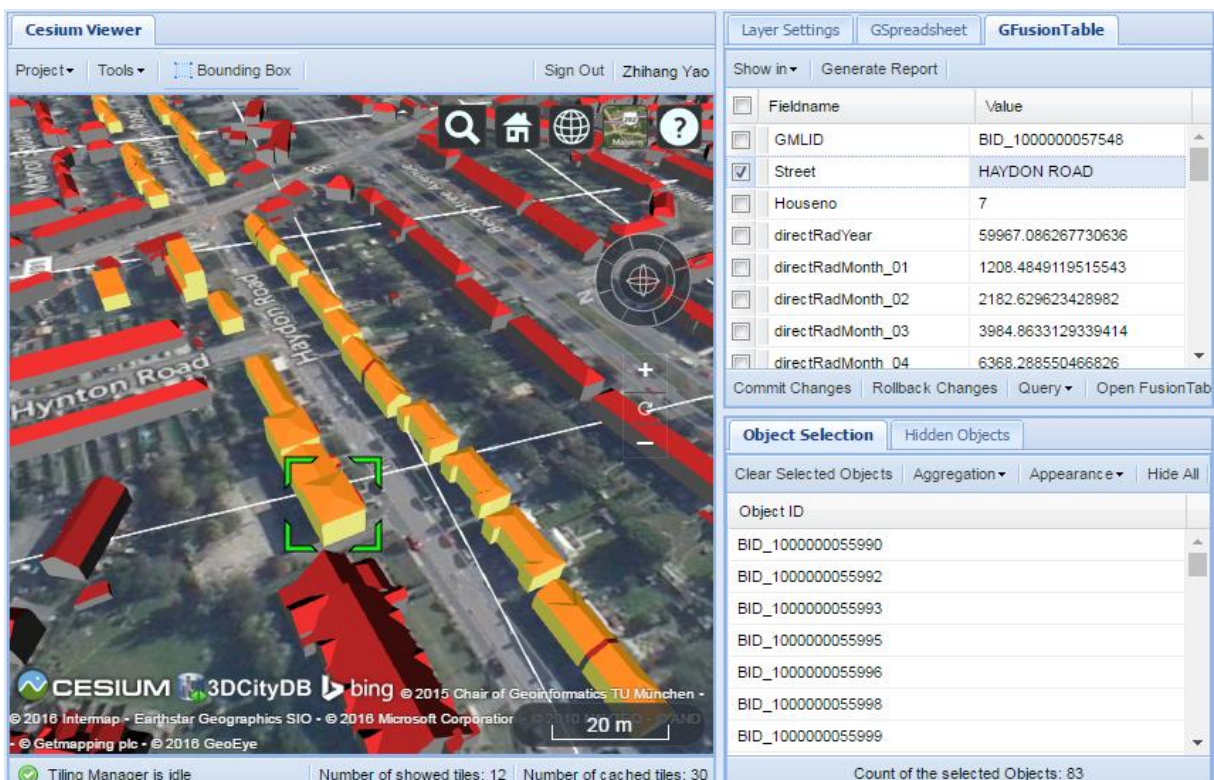


Figure 111: Example of performing a simple query based on an attribute



It is also possible to narrow down the query criteria by adding a further search condition. For example, a user may want to find out the buildings which are along “**HAYDON ROAD**” and the value of their yearly average direct solar irradiation is larger than **50000 kWh**. The user just needs to select the corresponding attributes by activating their check boxes, and the specified query criteria are combined in a logical “AND” operation (cf. Figure 112). If none of the checkboxes has been activated, no query will be applied.

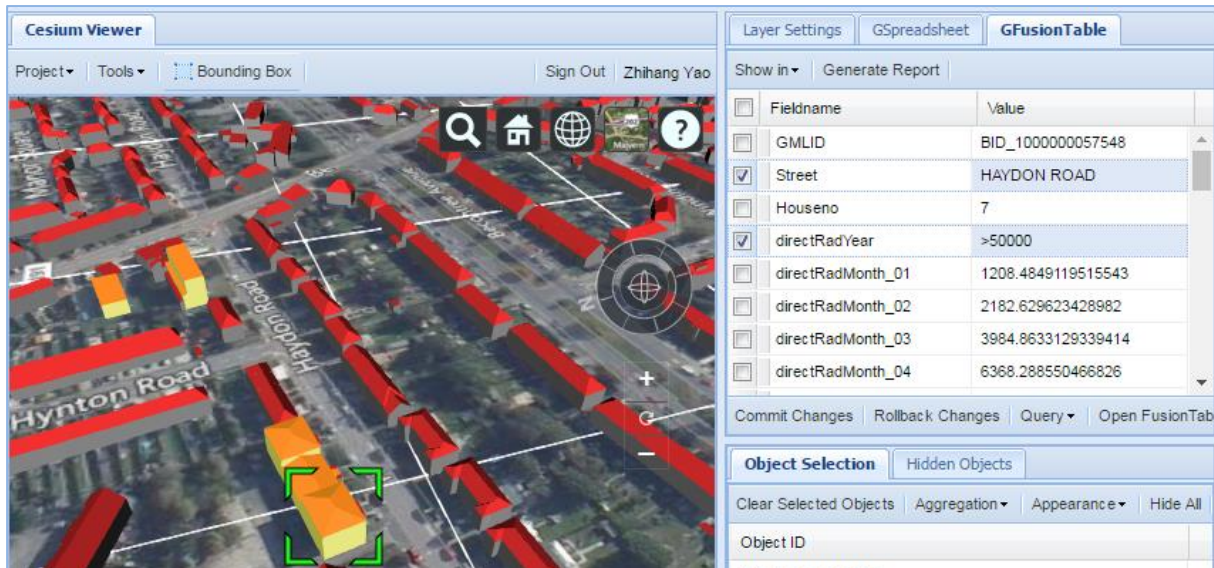


Figure 112: Example of performing a complex query based on multiple attributes

Aggregation functions like sum, average, min, and max are also supported for calculating the relevant statistic values (cf. Figure 113).

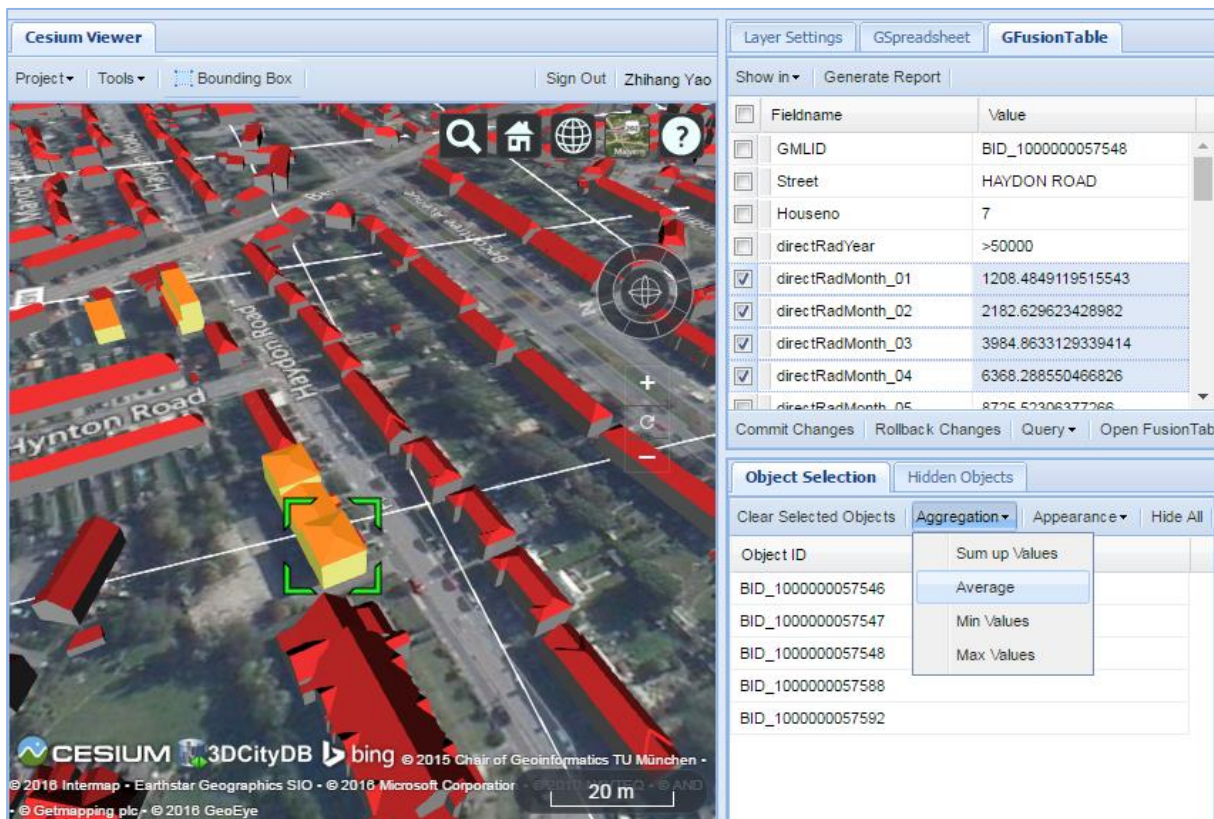


Figure 113: Example of performing aggregation calculation on multiple numeric attributes

A comprehensive report for summarizing the latest working and analysis status of the 3D web client can be created and displayed in a new browser window (cf. Figure 114). The created report especially lists all the selected attribute contents of the selected building objects as well as the calculation results of the aggregation functions. A screenshot of the current 3D map view can also be printed in the report.


Project - London_LBBD_Solar_Demo					
 3DCityDB Webclient V2.0 ©2012-2016 Chair of Geoinformatics TU München					
Report generated by: Zhihang Yao					
Generated Date: 2016-4-27 19:10:51					
Project URL: <a href="http://www.3dcitydb.org/3dcitydb-web-map-tum/2.0/index.html?&amp;config=https%3A%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1LJzTZJnscoyPL7YrSiMvvDLrKoQsR41A9vXd-9WzXQ%2Fedit%3Fusp%3Ddrivesdk&amp;latitude=51.55818316164792&amp;longitude=0.12093069453319732&amp;height=161.90681441169963&amp;h">http://www.3dcitydb.org/3dcitydb-web-map-tum/2.0/index.html?&amp;config=https%3A%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1LJzTZJnscoyPL7YrSiMvvDLrKoQsR41A9vXd-9WzXQ%2Fedit%3Fusp%3Ddrivesdk&amp;latitude=51.55818316164792&amp;longitude=0.12093069453319732&amp;height=161.90681441169963&amp;h</a>					
Active layer: London_LBBD_Buildings					
Object Selection: 'Street'='HAYDON ROAD' and 'directRadYear'>50000					
Number	GMLID	directRadMonth_01	directRadMonth_02	directRadMonth_03	directRadMonth_04
1	BID_1000000057546	1199.0403964781344	2165.1858541267607	3954.309775170164	6322.23598917861
2	BID_1000000057547	1150.8184432793696	1974.8480624260674	3579.5983729421937	5668.72659333612
3	BID_1000000057548	1208.4849119515543	2182.629623428982	3984.8633129339414	6368.288550466826
4	BID_1000000057588	1369.4600604067928	2694.59026996478	4963.736672721029	7992.537904023674
5	BID_1000000057592	1270.6594316439011	2625.869067599025	4833.475895643853	7732.886197493827
Aggregate Functions		directRadMonth_01	directRadMonth_02	directRadMonth_03	directRadMonth_04
Sum	of 5 Objects	6198.463243759753	11643.122877545615	21315.984029411182	34084.67523449906
Average	of 5 Objects	1239.6926487519506	2328.624575509123	4263.196805882237	6816.935046899812
Max	of 5 Objects	1369.4600604067928	2694.59026996478	4963.736672721029	7992.537904023674
Min	of 5 Objects	1150.8184432793696	1974.8480624260674	3579.5983729421937	5668.72659333612

Figure 114: Example of the automatically generated report showing an overview of the statistic information

### 6.3.2 Use Case 2: Energy Atlas Berlin

In the context of the smart city project “Energy Atlas Berlin”, an analysis and decision support tool for strategic energy planning at the city and district scale has been developed. It is based on the semantic 3D city model of Berlin, which was modeled and represented according to the CityGML standard. The Energy Atlas Berlin includes all data from the Solar Atlas Berlin including the rating of the suitability of all individual roof surfaces for each of the 550,000 buildings in Berlin for the production of photovoltaic and solar thermal energy. It also integrates the methods for the estimation of energy demand e.g. heating energy and electrical energy etc. and for the assessment of the energetic retrofitting possibilities on the individual building.

Based on the developed multi-level system and the 3D web client, an interactive energetic building retrofitting application has been developed which allows to assess the heating energy demand of every residential building (cf. Figure 115). This application allows stakeholders from housing companies, energy consulting firms, city administration as well as building owners and tenants to explore the energy demand of individual buildings or any group of buildings. Users can virtually apply retrofitting measures to the building(s) and directly see estimates for the potential savings of energy and money, as well as the estimated costs for



refurbishment. Nearly the entire computation logic has been implemented using formulas in a Cloud-based Google Spreadsheet (cf. Kaden 2014).

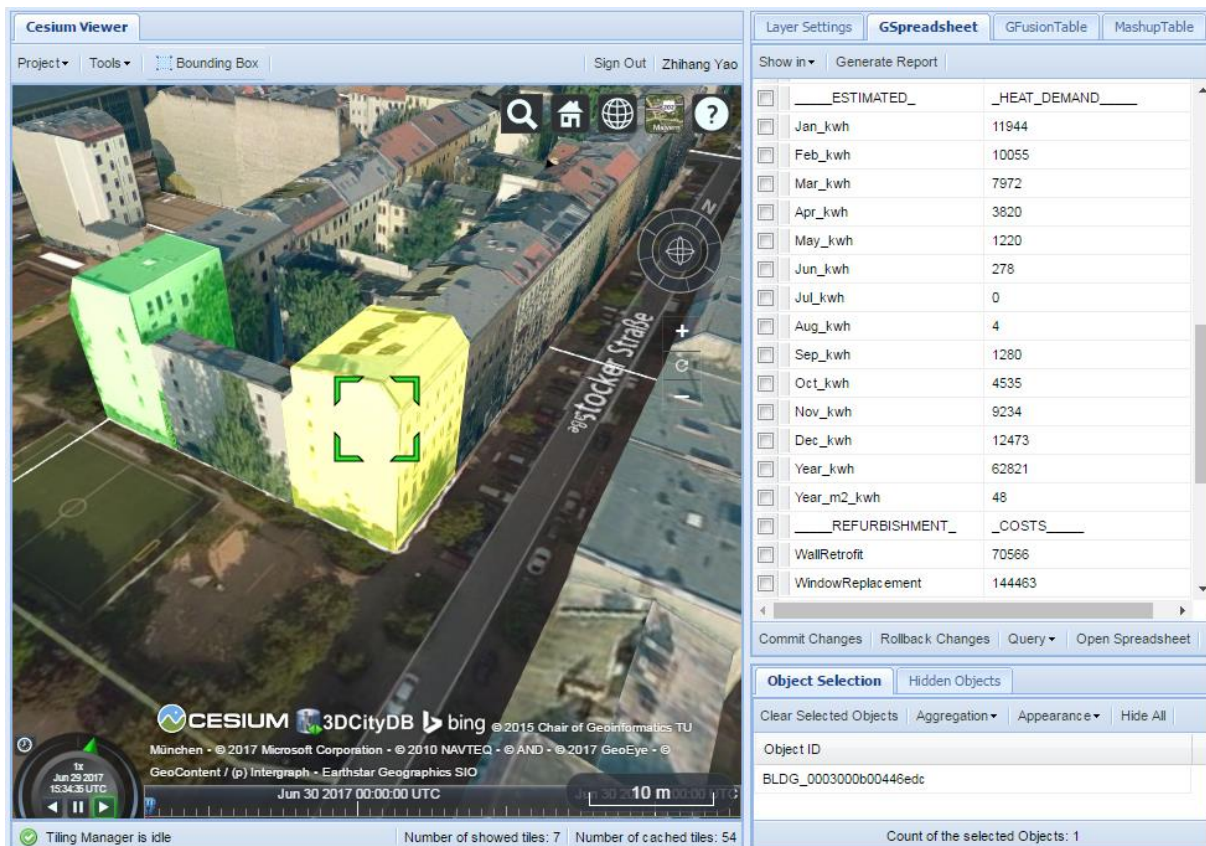


Figure 115: Example of showing the estimated energy demand for a selected building

Figure 116 shows an example of using the spreadsheet-based calculation engine to perform the ad-hoc simulation of the retrofitting measure. In this example, the heat transfer coefficients (called U values) of the outer walls, roofs and windows can be directly changed in the attribute table in the web client. The new U value entered by the user can be directly passed to the online spreadsheet to run the pre-defined formula for calculating the expected heat energy demand value of the annual heat demand per m<sup>2</sup> usable area. The respective calculation methods and formulas are described in detail in the work of (Kaden & Kolbe 2013, Kaden 2014).

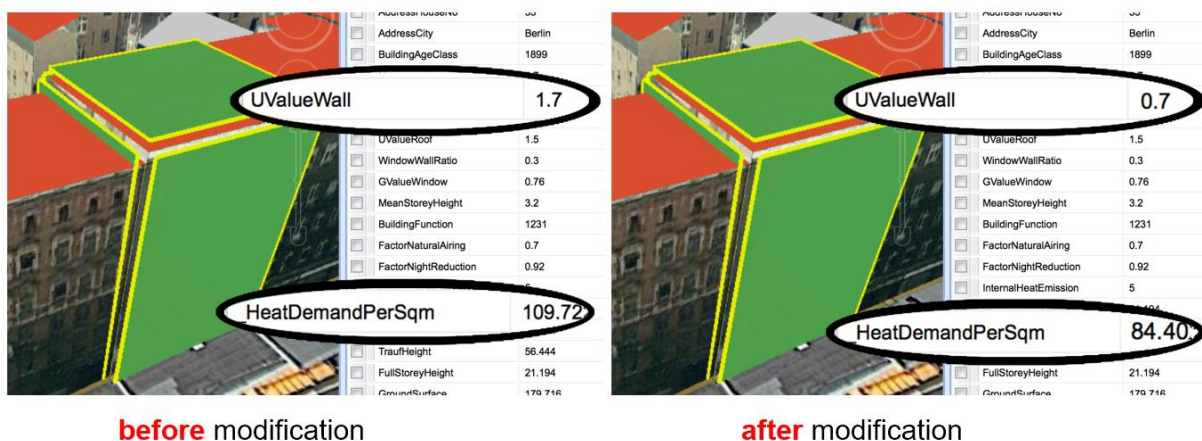


Figure 116: Ad-hoc estimation of the heating energy demand for one building (cf. Yao et al. 2014)



## Chapter 7 Discussion, Conclusions, and Outlook

In this chapter, the main results of this dissertation work including the developed methods and approaches are first summarized regarding the objectives and motivations of this thesis. In the second subsection, the major contributions of this thesis in the fields of scientific research, industrial applications, academic teaching, as well as international standardization are concluded. Finally, the relevant aspects about the future research are identified and outlined in the last subsection.

### 7.1 Discussion

In this section, the research and development results will be discussed by answering the questions and hypotheses stated in the first chapter.

#### **1) Which technologies and standards are important for realizing the efficient management, interaction, and visualization of semantic 3D city models using a computer system ?**

The first important technology are *Geographic Information Systems (GIS)*, which are a specific kind of application system designed for performing complex operations on geospatial data regarding the data acquisition, storage, management, analysis, and visualization. Since semantic 3D city models are a formal description of the heterogeneous geospatial data, the GIS technology provides the ideal platform to bridge the gap between the virtual urban information and the advanced computer applications on a variety of platforms or devices such as personal computers and mobile devices. Another key technology are *Spatial Database Management Systems (SDBS)*, which is one of the most important component in a GIS infrastructure to provide a powerful means for the efficient management of large 3D geospatial data. Concerning the high hardware costs of a desktop and server-based GIS infrastructure, the *Cloud Computing* technology paves a new way for an economical business mode which allows IT vendors to provide contracted services, such that the GIS users do not need to setup an in-house hardware infrastructure by their own. It also allows GIS companies or developers to not only minimize the resource and maintenance costs but also to speed up the development time for their business products. In addition, the *Geospatial Data Modelling* concept offers a systematic approach and standard like CityGML to harmonize the heterogeneous geospatial data into a common data model to facilitate the interoperable exchange of 3D city models between different application systems. For 3D visualization and interaction, the technologies *3D Graphics Visualization* and *Digital Virtual Globe* play the essential role, since they can offer the relevant functionalities for building a 3D user interface to realize high-performance exploration and 3D visualization of 3D city models on diverse platforms.

#### **2) What are the key aspects for realizing the efficient storage and management of large and complex-structured semantic 3D city models regarding the data modelling and software implementation ?**

The efficient management of semantic 3D city models in GIS applications requires a universal information model that should be able to establish a common definition of the complex city model objects, attributes, and relations along with a standardized exchange

format that can be interpreted by and exchanged between applications using a common interface. In this context, the international standard CityGML has already been developed which defines a rich feature catalogue and platform-independent UML model for the most relevant 3D topographic features like buildings, bridges, waters, and vegetation etc. according to well-defined international standards and specifications. In order to efficiently store CityGML datasets with large file sizes, the spatially-enhanced relational database systems are suitable means, which are able to make full use of their spatial capabilities and functions to handle 2D/3D spatial contents. Furthermore, the relational database schema must be carefully designed to create a compact table structure that allows to perform the operations reading, writing, querying, and updating etc. of large geospatial data efficiently. To realize this, a number of relational modelling concepts and approaches are identified which have also been successfully employed to develop an efficient geo-database in the framework of the 3DCityDB project. In addition, a complete 3DCityDB-compliant software toolkit provides the main functionalities supporting the work chain, starting with the reading, processing, and writing of the data contents in the database, via the conversion to different model representations, up to high-performance data visualization and exploration on a 3D mapping application.

### **3) How to implement the extensions to the semantic 3D city model according to the CityGML standard and then develop a dynamically extendable database for dealing with these extensions efficiently ?**

CityGML's extension mechanism ADE allows to dynamically extend the existing CityGML models by incorporating additional feature classes or extra attribute types into the CityGML framework. Such extensions can be maintained as a platform-independent information model using UML diagrams and can be implemented as an XML-based schema in compliance with GML schema through an automatic procedures according to the UML-to-XML encoding rules of the ISO 19136 standard. Concerning the data storage, the 3DCityDB database schema has already provided an advanced database implementation for supporting high-performance storage of standard CityGML data and the database must, hence, only become being extendable for attaching additional compact database schemas to support the handling of arbitrary CityGML ADE datasets with complex data model structures as well. While the XML schema of a CityGML ADE natively represents an object-oriented data structure, the target database schema can be generated from a dedicated model transformation process to execute the conversion of an object-oriented model (input model) to a relational database model (output model). To realize this, both the input and output models have to be first mapped onto some particular kinds of computer-interpretable formats or representations. This way, model transformation processes can be automatically carried out by applying a set of user-defined mapping rules by means of a computer-aided transformation system to produce the individual 3DCityDB-compliant database schema for the respective CityGML ADE.

### **4) Compared to the existing solutions, is there any further advanced approach that allows the automatic derivation of a compact relational database model from a CityGML extension which is structured as a complex object-oriented data model?**

According to the survey of the existing model transformation solutions, it was identified that most of them have very limited abilities in generating compact relational structures, because the employed mapping rules of these software tools are too rigid and can hence easily result in

a large amount of database tables, which will cause time-consuming issues when, for example, performing a complex query on those data contents that are distributed over many database tables linked with relational joins. In addition, the transformation rules are normally hard-coded within the software lacking the flexibility in extending or introducing additional new rules. Thus, it would be good to find a new way that allows developers to formally represent the mapping rules learned from the design decisions of the 3DCityDB to also automatically derive compact database schemas for arbitrary CityGML ADEs. To overcome these issues, both model representations can be fully represented using proper graph structures comprising a set of typed and attributed graph nodes and arcs which can represent the model entities and their interrelationships respectively. In addition, the mapping relationships between the model objects e.g., classes and tables can also be declaratively expressed using directed graph arcs to connect the respective graph nodes. This way, the model transformation problem can be completely expressed as a graph transformation problem which can be solved using *Graph Transformation Systems* owning the following advantages:

- In a graph transformation system, the initial state of the model transformation process can be formulated as a typed attributed graph called “host graph”. The entire process can be carried out by applying a set of so-called *graph transformation rules* which can be declaratively defined by users to reflect the logical mapping of the source model onto the target one. Each rule allows, according to the various transformation conditions, to rewrite the host graph in a way of substituting a subgraph by a new one for executing a corresponding calculation step. Moreover, a group of graph transformation rules can also be combined to perform a more complex transformation procedure. This way, the results including the desired relational database model as well as the mapping relationships between the both model representations can be yielded by interpreting the processed host graph once the graph transformation has been completed.
- Some graph transformation systems like AGG provide a very powerful control-flow mechanism called *layered-based* transformation which is an important concept for handling conflicting graph transformation rules, which can occur in case that multiple rules are applicable at the same time. The general idea of such layer-concept is to schedule the processing sequence of all transformation rules by grouping them into a set of numbered layers which can be sorted in a descending order based on the user-defined execution priorities and as such will be applied successively. In this way, each layer will have a small number of application rules whose potential conflicts can be easily determined and avoided by decomposing the conflicting rules into further layers. This mechanism is very essential for model transformations since it can ensure the proper termination of the transformation processes that have all resolved conflicting rules.
- The concept of the *type graph* coming with the graph transformation system like AGG allows the formal definition of the meta-model of all graph elements in the graph transformation system and can be used as a global constraint allowing to guarantee the structural and semantic consistency of the host graph and transformation rules throughout the transformation process. It is hence possible to design the meta-graph

for the object-oriented model and relational database model as well as for the mapping relationship between the individual model objects, e.g. the mapping relationship between classes and tables. The structure of the meta-graph can be conceptually expressed as a UML-diagram which can later be directly mapped onto the corresponding graph representation.

Based on the above-described features of graph transformation systems, a graph-based approach has been successfully developed for deriving compact relational database schemas from CityGML ADEs.

### **5) How to realize the efficient visualization of large semantic 3D city models along with their extensions in a web browser, where users are able to explore the data model information interactively ?**

The semantic 3D city model standard CityGML was primarily designed as a universal information model which is able to carry the heterogeneous urban information in an XML-based data structure. It is however not very suitable for the purpose of 3D visualization and shall be converted to specific 3D visualization models with appropriate spatial tiling structure and coordinate reference system, so that 3D mapping applications are able to dynamically load the relevant data tiles according to the current camera perspective for the efficient rendering of 3D models during runtime. With the growing capabilities of modern web browsers and the consequent advancement in HTML5 and WebGL-based 3D virtual globes, the web-based 3D visualization of large spatial data on most major operating systems and platforms became possible. Another aspect regarding the web-based exploration of semantic 3D city models is the linking with and quering of thematic information in addition to the pure 3D visualization. The proposed solution in the thesis is to utilize Cloud-based online spreadsheets for storing the thematic information which can be provided by third parties or outsourced from the central database. Consequently, the Cloud-based online spreadsheet can be used as an intermediate data container for keeping the contents up-to-date without affecting the original 3D city models stored in the central database. This way, any update of thematic contents will exclusively take place within the online spreadsheet and can also be written back to the database at any time. In addition, based on the unique identifier of each city model object, multiple online spreadsheets containing different domain-specific information can be linked together and displayed to users simultaneously. Furthermore, the access of online spreadsheets can also be easily controlled via Cloud services. For example, once an online spreadsheet has been created by a user, it can be quickly shared with other users by grouping them with different access rights e.g. read-only or the full access with write privilege.

***Hypothesis:* It is possible to develop such an application system that supports management, visualization, and interaction of extendable semantic 3D city models, and at the same time reaches a good balance between the high model complexity and specific user needs. This balance can be evaluated against the criterion, whether end users who are not GIS experts can also access complex 3D city models and accomplish various domain-specific analysis and simulation tasks.**

In the context of Smart City projects, the growing complexity of the application scenarios strongly requires a comprehensive and integrative platform which allows for facilitating a range of analysis, simulation, and modification operations on the complex-structured 3D city

models. Since most end users from specific application domains are not GIS experts, it hence makes sense to make the system platform scalable such that a wide group of application users are able to easily access and handle the 3D city models during the planning, design, and development process. To reach this goal, a new multi-level system for building up such kind of GIS environments has been proposed in this dissertation work and also been successfully implemented based on existing software tools and 3D city model standard. It introduces an additional application level that bridges the gap between the end users and the complex semantic 3D city models stored in a central database which maintains the heterogeneous urban information according to the CityGML standard along with its ADE mechanism. The technical implementation of the application level has been realized based on the so-called *App-concept*, which defines that for specific users or user groups, each application should have a very limited range of functions and must hence be relatively easy to learn and intuitive to operate. Based on this concept, the required 3D city models of a specific app shall be simplified by extracting them from the central database according to the thematic and spatial filter criteria in order to generate a relatively small 3D visualization model whose related thematic information can be mapped onto a simple table structure. As an intuitive graphical user interface, a 3D web client has been developed whose functionalities go beyond the 3D visualization and exploration and allows for performing various analysis and simulations on 3D city models using the Cloud Computing technology. For example, it is possible use Cloud-based online spreadsheets to develop a light-weight calculation engine by defining a spreadsheet formula on a column whose record values can be automatically computed from the values of the other columns. Any ad-hoc changes to the values of these columns can be made by users through the web client and the calculation results will be immediately displayed in response to the user operations. In this way, users are shielded from the complex 3D city models and only need to focus on their business tasks by accessing the simplified data and operating the analytic tools.

## 7.2 Contribution of the Thesis

The key contributions of this dissertation work for a range of fields are summarized as follows.

### 1) Contributions to scientific research

One of the main contributions of the thesis is the comprehensive review of the relevant standards and technologies that are of great importance for the efficient management, interaction, and visualization of 3D geospatial data. It especially provides GIS researchers with the fundamental basis for the systematic development of a domain-extendable semantic 3D city model according to the international standards and summarizes the key solutions for the creation of a compact database schema using spatially-enhanced relational database management systems. In addition, the conceptualization of the logical mapping rules for the model transformation facilitates the development of a new graph-based approach which go beyond existing solutions for the automatic derivation of relational database schemas and which can provide the GIS researcher with a new idea for the transformation between different types of application schemas i.e. JSON, XML, and graph database etc. Moreover, the implementation results obtained from the thesis also strongly shows the possibility of utilizing graph transformation systems to accomplish the complex model transformations in



the geospatial domain. Thus, it is reasonable to believe that the developed methods and concepts can be adopted and applied for other data models like INSPRE, IndoorGML, and IFC etc. in future research.

## **2) Contribution to practical application**

During the research of this thesis, a number of software tools have been developed or extended for improving the handling of 3D city models for both CityGML 1.0 and 2.0. Most of these software tools are completely open source and freely available for the public. For example, starting from version 3.3.0 of the 3DCityDB, the developed 3D web client (cf. 4.3.4) has been introduced into the 3DCityDB toolkit to serve as a web-based front-end for high-performance 3D visualization and interactive exploration of arbitrarily large semantic 3D city models using Cloud technology. Also, many consequent extensions have been made to the CityGML Import/Export tool in order to generate pre-styled 3D computer graphics models with a simple tiling structure for 3D geo-visualization. To date, the developed software tools have been successfully employed in many research projects, and a number of companies worldwide are using the software at the core of their commercial products to process semantic 3D city models in a range of applications and infrastructures. At the time of writing this thesis, a new version of the 3DCityDB (cf. Chapter 4) has been completed which includes the support of CityGML ADEs and will be released in the year 2019.

## **3) Contributions to academic teaching and practical training**

The presented work provides a comprehensive introduction to the relevant concepts and workflow ranging from model-driven based 3D geospatial data modelling, via the relational database modelling, up to the 3D geo-visualization and exploration on the web. It is based on state-of-the-art technologies and international standards along with numerous application examples, which together forms an ideal and extensive lecture roadmap for GIS students to help them in learning the advanced knowledge around the topics of ‘*Applied Geoinformatics*’. In addition, the key concepts of the international OGC standard CityGML as well as its extension mechanism have also been explained in detail which covers the fundamental aspects for the modelling of an extendable semantic 3D city model as well as for its utilization in practical applications. Also the illustrated technical details about the software implementation of the individual 3DCityDB software tool can substantially help GIS developers and software vendors to get an easy start with the understanding of the software structures and allow them to rapidly extend the existing functionalities of the software modules according to the specific project needs. Moreover, for those 3DCityDB users who are not experienced in software programming, the author of this thesis has written a hands-on tutorial<sup>6</sup> which simplifies the details of the software implementation and gives a step-by-step instructions on how to use the CityGML and 3DCityDB to build a customized 3D web application.

## **4) Contributions to international standardization**

The current CityGML specification lacks a reasonable definition of the scope of CityGML ADEs whose feature classes and data types can, in principle, be defined as any complex types in XML schemas. However, as stated in the CityGML specification, new feature classes of an

---

<sup>6</sup> Tutorial available at: <https://www.gis.bgu.tum.de/en/projects/3dcitydb/>

ADE shall be defined based on the extension of the existing CityGML classes and this rule was rarely respected in practice as many feature classes in the existing ADEs like Energy-ADE and UtilityNetwork-ADE etc. have been directly derived from the GML feature class instead. Besides, it also lacks extensive instructions about the systematic development of an ADE regarding its UML data modelling as well as XML encodings which have only been addressed later by (van den Brink et al. 2013). Against this background, this thesis offers a complete framework to incorporate the existing model-driven approaches for the development of ADEs with respect to the ISO 19136 standard which stipulates that the ADE classes must be derived from either the GML or CityGML classes. In addition, the developed framework also introduces a systematic way for the automatic derivation of a CityGML-compliant relational database schema from a valid ADE application schema. The applicability of this framework has been successfully evaluated based on an artificial ADE which is representative for most ADEs in practice. Above, the implementation results around this artificial ADE including its UML models, XML and relational database schema, as well as the corresponding schema mapping file can serve as a best practice for the future CityGML specification.

### 7.3 Outlook and Future Research

Future work should mainly focus on the continuing improvement of the presented graph-based approach by fine-graining the developed graph transformation rules in order to derive the desired database schemas according to the various optimization concepts, constraints, and principles of model transformation. This would allow to abstract a set of declarative graph transformation rules from the design decision of the 3DCityDB database schema which can then be automatically derived from the original CityGML's XML schema definition files or UML models. This way, the 3DCityDB developers will be able to efficiently maintain the database schema at an abstract conceptual level from which the SQL definitions of different spatial database products e.g. Oracle Spatial, PostgreSQL/PostGIS, and SAP HANA can be automatically generated. Another positive impact is that the 3DCityDB can also be rapidly updated in response to any changes or upgrades made to the CityGML data models without needing to modify the database schema manually. Moreover, the long-term goal is to make the 3DCityDB far more generic for translating arbitrary GML application schema to efficient database schemas automatically.

The current approach for the database import and export of CityGML ADE datasets is based on a plugin-like solution which requires quite a lot of manual implementation on the import/export functionalities for each individual ADE. This causes high costs in terms of programming skills and investments in case of complex ADEs and therefore raises the demand for a generic solution that should allow to automatically read and write CityGML ADE instance documents without needing to deploy additional plugins. This concept could also be applied to other 3DCityDB software tools like KML/COLLADA/gITF Exporter and the Spreadsheet Generator Plugin in order to support the automatic creation of 3D visualization models and Cloud-based online spreadsheets for arbitrary ADEs which would be easily accessed and explored using the developed 3D web client. With these improvements, the 3DCityDB along with its toolkit will become an extendible platform with the generic

support for the interoperable management, access, analysis, visualization, and exploration of arbitrary GML data.

As mentioned in section 5.2.1, it is possible to represent the aggregation structure of a 3D city object e.g. building using the glTF-based batched 3D models on a web client. With this approach, the currently existing 3D mapping applications provide two interaction modes which allow users to select the building object as a whole or click on its subdivided parts like wall and roof surface to query the related thematic information (cf. Schilling et al. 2016). The two modes are mutually exclusive and must be manually switched by users in order to interact with the objects on different construction levels. This operation behavior is however not very friendly for the users and can be improved with better interaction functionalities in future. For example, once a building object has been firstly clicked, its root element along with its subdivided parts shall be identified and highlighted. With the second click on the same screen position, the respective sub-part like wall or roof surface will be selected and all the rest sub-parts should be unhighlighted. When clicking on the same building object once again, the element like window or door at a deeper level shall be selected. Depending on the selected object, the associated thematic information should be respectively displayed in a table view as usual, besides which the structure information about the decomposition hierarchy could also be outlined on a dialog window using a tree-like structure which can be freely navigated to select the individual building elements at different hierarchy levels. The possibility and usability of such kinds of interactions on the Web will have to be proven in the future research work.

## Bibliography

- AGG (2006) The AGG 1.5.0 Development Environment - The User Manual. <http://www.user.tu-berlin.de/o.runge/agg/AGG-ShortManual/AGG-ShortManual.html>. Accessed 13 Nov 2018.
- Agoub A, Kunde F, Kada M (2016) Potential of Graph Databases in Representing and Enriching Standardized Geodata. In: Kersten TP (ed) Tagungsband der 36. Wissenschaftlich-Technischen Jahrestagung der DGPF in Bern. Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. (DGPF), vol 25, pp 208-216.
- Agugiario G, Benner J, Cipriano P, Nouvel R (2018) The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations. *Open Geospatial Data, Software and Standards*. 3(2).
- Almendros-Jiménez JM, Becerra-Terón A, García-García F (2010) Development of a Query Language for GML based on XPath. In: Kovács L, Kutsia T (eds) Proceedings of 6th International Workshop on Automated Specification and Verification of Web Systems, 30-31 Juli 2010, Wien, Austria. EPiC Series in Computing 18, EasyChair 2013, pp 51-64.
- Altmaier A, Kolbe TH (2003) Applications and Solutions for Interoperable 3d Geo-Visualization. In: Fritsch D (ed) Proceedings of the photogrammetric week 2003 in Stuttgart. Wichmann Verlag, Heidelberg, pp 251-267.
- Ambler SW (1997) Mapping objects to relational databases: What you need to know and why. <http://caminotics.ort.edu.uy/innovaportal/file/2032/1/mappingobjectstorelationaldatabases.pdf>. Accessed 13 Nov 2018.
- Amer-Yahia S, Du F, Freire J (2004) A Comprehensive Solution to the XML-to-Relational Mapping Problem. In: Proceedings of the 6th annual ACM international workshop on Web information and data management, 12-13 November 2004, Washington DC, USA, pp 31-38.
- Amirian P, Alesheikh A (2008) A Service Oriented Framework for Disseminating Geospatial Data to Mobile, Desktop and Web Clients. *World Applied 32 Sciences Journal*. 2008;3(1):140-153.
- Autodesk (2014) Autodesk FBX SDK Documentation. <http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/>. Accessed 13 Nov 2018.
- Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010). A View of Cloud Computing. *Communications of the ACM*. 53(4):50-58.
- Atkinson M, Bancilhon F, DeWitt D, Dittrich K, Maier D, Zdonik S (1992) The Object-Oriented Database System Manifesto. In: Bancilhon F, Delobel C, Kanellakis P (eds) Building an Object-oriented Database System, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1-20.
- Awang MK, Labadu NL (2012) Transforming Object Oriented Data Model to Relational Data Model. *International Journal on New Computer Architectures and Their Applications (IJNCAA)*. 2(3):402-409.
- Alattas A, van Oosterom PJM, Zlatanova S (2018) Deriving the Technical Model for the Indoor Navigation Prototype based on the Integration of IndoorGML and LADM Conceptual Model. In: Proceedings of the 7th Land Administration Domain Model Workshop, 12-13 April 2018, Zagreb, Croatia.
- Barnes M, Finch EL (2008) COLLADA – Digital Asset Schema Release 1.5.0. Specification. [https://www.khronos.org/files/collada\\_1\\_5\\_release\\_notes.pdf](https://www.khronos.org/files/collada_1_5_release_notes.pdf). Accessed 13 Nov 2018.

- Beauregard B, Murray C, Speckhard B (2009). Oracle Database 11g Workspace Manager Overview. <https://www.oracle.com/technetwork/database/twp-appdev-workspace-manager-11g-128289.pdf>. Accessed 13 Nov 2018.
- Becker T, Nagel C, Kolbe TH (2013) Semantic 3D modeling of multi-utility networks in cities for analysis and 3D visualization. In: Pouliot J, Daniel S, Hubert F, Zamyadi A (eds) Progress and new trends in 3D Geoinformation sciences. Lecture notes in Geoinformation and cartography. Springer Berlin Heidelberg, pp 41-62.
- Beil C, Kolbe TH (2017) CityGML and the streets of New York - A proposal for detailed street space modelling. In: Kalantari M, Rajabifard A (eds) Proceedings of the 12th International 3D GeoInfo Conference 2017. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-4/W5. ISPRS, pp 9-16.
- Bhardwaj S, Jain L, Jain S (2010) Cloud computing: A study of infrastructure as a service (IAAS). International Journal of Engineering and Information Technology. 2(1): 60-63.
- Bhat MA, Shah RM, Ahmad B (2011) Cloud Computing: A solution to Geographical Information Systems (GIS). International Journal on Computer Science and Engineering (IJCSSE). 3(2):594-600.
- Biljecki F, Ledoux H, Stoter J (2016) An improved LOD specification for 3D building models. Computers, Environment and Urban Systems. 59:25-37.
- Biljecki F, Stoter J, Ledoux H, Zlatanova S, Çöltekin A (2015) Applications of 3D City Models: State of the Art Review. ISPRS International Journal of Geo-Information. 4(4):2842-2889.
- Biljecki F, Kumar K, Nagel C (2018) CityGML Application Domain Extension (ADE): overview of developments. Open Geospatial Data, Software and Standards. 3(13).
- Bohannon P, Freire J, Roy P, Siméon J (2002) From XML schema to relations: A cost-based approach to XML storage. In: Agrawal R, Dittrich K, Ngu AHH (eds) Proceedings of 18th International conference on data engineering, 26 February - 1 March 2002, San Jose, California, USA, pp 64-75.
- Boicea A, Radulescu F, Agapin LI (2012) MongoDB vs Oracle - database comparison. In: Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, Bucharest, Romania, 19-21 September 2012, pp 330-335.
- Bottoni P, Koch M, Parisi-Prsicce F, Taentzer G (2005) Termination of High-Level Replacement Units with Application to Model Transformation. In: Minas M (ed) Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM 2004), 30-30 September 2004, Rome, Italy. Electronic Notes in Theoretical Computer Science vol 127, issue 4. Elsevier, pp 71-86.
- Brinkhoff T (2005) Geodatenbanksysteme in Theorie und Praxis. Wichmann, Heidelberg.
- Büttner F, Gogolla M (2004) Realizing UML metamodel transformations with AGG. In: Heckel R (ed) Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004), 27-28 March 2004, Barcelona, Spain. Electronic Notes in Theoretical Computer Science, vol 109. Elsevier, pp 31-42.
- Buyya R, Yeo CS, Venugopal S (2008) Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: Proceedings of 10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008, 25-27 September 2008, Dalian, China, pp 5-13.
- Burggraf D (2015) OGC KML 2.3, Version 1.0, OGC Doc No. 12-007r2. Open Geospatial Consortium. <http://docs.opengeospatial.org/is/12-007r2/12-007r2.html> Accessed 13 Nov 2018.

- Chang KT (2006). Introduction to Geographic Information Systems. McGraw-Hill Education, New York City, USA.
- Chaturvedi K, Kolbe TH (2016) Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities. In: Dimopoulou E, van Oosterom PJM (eds) Proceedings of the 11th International 3D Geoinfo Conference. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-2/W1. ISPRS, pp 31-38.
- Chaturvedi K, Willenborg B, Sindram M, Kolbe TH (2017) Solar Potential Analysis and Integration of the Time-dependent Simulation Results for Semantic 3D City Models Using Dynamizers. In: Kalantari M, Rajabifard A (eds) Proceedings of the 12th International 3D GeoInfo Conference 2017. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-4/W5. ISPRS, pp 25-32.
- Chaturvedi K, Yao Z, Kolbe TH (2015) Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL. In: Kersten TP (ed) Tagungsband der 35. Wissenschaftlich-Technischen Jahrestagung der DGPF, 16-18 March 2015, Köln. Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. (DGPF), vol 24, pp 296-306.
- CesiumJS (2019) Open Source 3D Mapping. <https://cesium.com/cesiumjs/>. Accessed 13 Jan 2019.
- Christen M (2016) Openwebglobe 2: Visualization of Complex 3D-Geodata in the (mobile) Web-browser. In: Halounova L, Schindler K, Limpouch A, Pajdla T, Šafář V, Mayer H, Oude Elberink S, Mallet C, Rottensteiner F, Brédif M, Skaloud J, Stilla U (eds) Proceedings of XXIII ISPRS Congress, 12–19 July 2016, Prague, Czech Republic. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences vol III-3. ISPRS, pp 401-406.
- Chuan CH (2012) 3D Graphics with OpenGL - Basic Theory. [http://www.ntu.edu.sg/home/ehchua/programming/opengl/cg\\_basicstheory.html](http://www.ntu.edu.sg/home/ehchua/programming/opengl/cg_basicstheory.html). Accessed 13 Nov 2018.
- Coffman EG, Garey MR, Johnson DS, Tarjan RE (1980) Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*. 9(4):808-826.
- Cozzi P (2015) Graphics Tech in Cesium - Renderer Architecture. <https://cesiumjs.org/2015/05/15/Graphics-Tech-in-Cesium-Architecture/> Accessed 13 Nov 2018.
- Cozzi P, Lilley S, Getz G (2019) 3D-Tiles Specification, Version 1.0, OGC Doc No. 18-053r2. Open Geospatial Consortium. <https://portal.opengeospatial.org/files/18-053r2> Accessed 31 Jan 2019.
- Cozzi P, Ring K (2011) 3D Engine Design for Virtual Globes (1st edition). CRC Press, Boca Raton, Florida.
- de Laat R, van Berlo L (2010) Integration of BIM and GIS: The Development of the CityGML GeoBIM Extension. In: Kolbe TH, König G, Nagel C (eds) Advances in 3D Geo-Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer, pp 211-225.
- Deegree (2017) deegree webservice 3.3.13 documentation - Feature Stores <http://download.deegree.org/documentation/3.3.13/html/featurestores.html#anchor-mapping-wizard>. Accessed 13 Nov 2018.
- Döllner J, Kolbe TH, Liecke F, Sgouros T, Teichmann K (2006) The Virtual 3D City Model of Berlin- Managing, Integrating, and Communicating Complex Urban Information. In: Proceedings of the 25th International Symposium on Urban Data Management UDMS, 15-17 May 2006, Aalborg, Denmark.

- Ehrig H, Ehrig K, de Lara J, Taentzer G, Varró D, Varró-Gyapay S (2005) Termination Criteria for Model Transformation. In: Cerioli M (ed) *Fundamental Approaches to Software Engineering. Proceedings of the 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS, 4-8 April 2005, Edinburgh, UK. Lecture Notes in Computer Science*. Springer, pp 49-63.
- Ehrig H, Ermel C, Golas U, Hermann F (2015) *Graph and Model Transformation - General Framework and Applications*. Springer.
- Emgård L, Zlatanova S (2008) Implementation alternatives for an integrated 3D Information Model. In: van Oosterom PJM, Zlatanova S, Peninga F, Fendel EM (eds) *Advances in 3D Geoinformation Systems*. Springer, pp 313-329.
- ESRI (2017) ArcGIS Online Help. <http://doc.arcgis.com/en/arcgis-online/> Accessed 13 Nov 2018.
- Evans B, Sabel CE (2012) Open-Source web-based geographical information system for health exposure assessment. *International Journal of Health Geographics*. 11(2).
- Faucher C, Lafaye JY (2007) Model Driven Engineering for implementing the ISO 19100 series of international standards. In: *Proceedings of CoastGIS 07, the 8th International Symposium on GIS and Computer Mapping for Coastal Zone Management, 8-10 October 2007, Santander, Espagne*, pp 424-433.
- Fernando N, Loke SW, Rahayu W (2013) Mobile cloud computing: A survey. *Future Generation Computer Systems*. 29(1):84-106.
- Florescu D, Kossmann D (1999) Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin (IEEE-CS)*. 22(3):27-34.
- Foley JD, van Dam A, Feiner SK, Hughes J (1995) *Computer Graphics: Principles and Practice (2nd edition)*. Addison-Wesley, Boston, Massachusetts, United States.
- Folli A, Mens T (2008) Refactoring of UML models using AGG. In: Margaria T, Padberg J, Taentzer G, Mens T, Van Paesschen E, Mens K, D'Hondt M (eds) *Proceedings of the Third International ERCIM Symposium on Software Evolution, 5 October 2007, Paris. Electronic Communications of the EASST vol 8*.
- Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I (2009) *Above the Clouds: A Berkeley View of Cloud Computing*. Technical report of Electrical Engineering and Computer Sciences University of California at Berkeley. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>. Accessed 13 Nov 2018.
- Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Gasevic D, Djuric D, Deved V, Selic B (2006). *Model Driven Architecture and Ontology Development*. Springer, New York, Inc., Secaucus, NJ, USA.
- Geiß R, Batz GV, Grund D, Hack S, Szalkowski A (2006) GrGen: A Fast SPO-Based Graph Rewriting Tool. In: Corradini A, Ehrig H, Montanari U, Ribeiro L, Rozenberg G (eds) *Graph Transformations. Proceedings of Third International Conference, ICGT 2006 Natal, 17-23 September 2006, Rio Grande do Norte, Brazil. Lecture Notes in Computer Science, vol 4178*. Springer, pp 383-397.
- Godsil C, Royle GF (2013) *Algebraic Graph Theory*. Springer, New York City, USA.
- Golobisky MF, Vecchietti A (2011) Fundamentals for the Automation of Object-Relational Database Design. *IJCSI International Journal of Computer Science*. 8(3):1694-0814.



- Gonzalez H, Halevy A, Jensen C, Langen A, Madhavan J, Shapley R, Shen W (2010) Google Fusion Tables: Data Management, Integration, and Collaboration in the Cloud. In: Hellerstein JM, Chaudhuri S, Rosenblum M (eds) Proceedings of the 1st ACM symposium on Cloud computing, 10-11 June 2010 Indianapolis, USA. ACM, pp 175-180.
- Google Elevation API (2017) <https://developers.google.com/maps/documentation/elevation/start>. Accessed 13 Nov 2018.
- Grossner K, Goodchild MF, Clarke K (2008) Defining a Digital Earth System. *Transactions in GIS* 12(1):145-160.
- Gröger G, Kolbe TH, Nagel C, Häfele KH (2012) OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0, OGC Doc No. 12 – 019. Open Geospatial Consortium. [https://portal.opengeospatial.org/files/?artifact\\_id=47842](https://portal.opengeospatial.org/files/?artifact_id=47842). Accessed 13 Nov 2018.
- Gaurkhede PP, Pursani PJ (2014) Survey of Object Oriented Database. *International Journal of Modern Trends in Engineering and Research*. 1(5):205-214.
- Güting RH (1994). An Introduction to Spatial Database Systems. *The VLDB Journal - The International Journal on Very Large Data Bases*. 3(4):357-399.
- Guttman A (1984) R-Trees - A Dynamic Index Structure for Spatial Searching. In: Yormark B (ed) Proceedings of the 1984 ACM SIGMOD international conference on Management of data, 18-21 June 1984, Boston, Massachusetts, USA. ACM Press, pp 47-57.
- Habel A, Müller J, Plump D (2001) Double-Pushout Graph Transformation Revisited. *Mathematical Structures in Computer Science*. 11(5):637-688.
- Herman L, Řezník T (2015) 3D Web Visualization of Environmental Information – Integration of Heterogeneous Data Sources When Providing Navigation and Interaction. In: Mallett C, Papanoditis C, Dowman I, Oude Elberink SJ, Raimond AM, Rotensteiner F, Yang M, Christophe S, Coltekin A, Bredif M (eds) Proceedings of ISPRS Geospatial Week 2015, 28 September - 3 October 2015, La Grande Motte, France. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol XL-3/W3. ISPRS, pp 479-385.
- Herreruella J, Nagel C, Kolbe TH (2012) Value-added Services for 3D City Models using Cloud Computing. In: Löwner MO, Hillen F, Wohlfart R (eds) *Geoinformatik 2012 "Mobilität und Umwelt"*, Konferenzband zur Tagung Geoinformatik 2012, 28-30 March 2012, Braunschweig. Shaker, pp 327-334.
- Hildebrandt D (2014) A Software Reference Architecture for Service-Oriented 3D Geovisualization Systems. *International Journal of Geo-Information (ISPRS)*. 3(4):1445-1490.
- Hofman W, Lohman W, Schelling A (2011). A Flexible IT Infrastructure for Integrated Urban Planning. *Journal of Theoretical and Applied Electronic Commerce Research*. 6(1):16-25.
- ISO (2003) ISO 19107:2003: Geographic information - Spatial schema.
- ISO (2005) ISO 19109:2005: Geographic information - Rules for application schema.
- ISO (2007) ISO 19111:2007: Geographic information - Spatial referencing by coordinates.
- ISO (2007) ISO 19136:2007: Geographic information - Geography Markup Language (GML).
- ISO (2011) ISO 19118:2011: Geographic information - Encoding.
- Jackson D (ed) (2014) WebGL Specification. <https://www.khronos.org/registry/webgl/specs/1.0/>. Accessed 13 Nov 2018.

- Jung V (2008) Integrierte Benutzerunterstützung für die Visualisierung in Geo-Informationssystemen. Dissertation, Technischen Universität Darmstadt.
- Haala N, Kada M (2010) An update on automatic 3D building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*. 65(6):570-580.
- Kaden R (2014) Berechnung der Energiebedarfe von Wohngebäuden und Modellierung energiebezogener Kennwerte auf der Basis semantischer 3D-Stadtmodelle. Dissertation, Technische Universität München.
- Kaden R, Kolbe TH (2013) City-Wide Total Energy Demand Estimation of Buildings using Semantic 3D City Models and Statistical Data. In: Isikdag U (ed) *Proceedings of the 8th International 3D GeoInfo Conference, 27-29 November 2013, Istanbul, Turkey*. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol II-2/W1. ISPRS, pp 163-171.
- Karimi HA, Roongpiboonsopit D, Wang H (2011) Exploring Real-Time Geoprocessing in Cloud Computing: Navigation Services Case Study. *Transactions in GIS*. 15(5):613-633.
- Keller W (1997) Mapping Objects to Tables - A Pattern Language. In: Buschmann F, Riehle D (eds) *Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany*. Siemens technical report 120/SW1/FB. pp 59-84.
- Kessenich J, Baldwin D, Rost R (2008) The opengl shading language.  
<http://www.cse.chalmers.se/edu/year/2010/course/TDA361/GLSLangSpec.Full.1.30.08.pdf>. Accessed 13 Nov 2018.
- Bhatia S, Cozzi P, Knyazev A, Parisi T (2017) glTF 2.0 - Runtime 3D Asset Delivery.  
<https://github.com/KhronosGroup/glTF/blob/master/specification/2.0/README.md>. Accessed 13 Nov 2018.
- Kolbe TH (2009) Representing and Exchanging 3D City Models with CityGML. In: Lee J, Zlatanova S (eds) *3D Geo-Information Sciences. Proceedings of the 3rd International Workshop on 3D Geo-Information, 13-14 November 2008, Seoul, Korea*. Lecture Notes in Geoinformation and Cartography. Springer, pp 15-31.
- Kolbe TH, Burger B, Cantzler B (2015) CityGML goes to Broadway. In: Fritsch D (ed) *Photogrammetric Week*. Wichmann, Stuttgart, pp 343-356.
- Kolbe TH, Steinrücken J, Plümer L (2003) Cooperative Public Web Maps. In: *Proceedings of the International Cartographic Congress (ICC), 10-16 August 2003, Durban, South Africa*. International Cartographic Association (ICA), pp 1062-1071.
- Kolbe TH, Yao Z, Nagel C, Redweik R, Willkomm P, Hudra G, Müftüoğlu A, Kunde F (2017) 3D Geodatabase for CityGML Documentation Version 3.3.0.  
[https://www.3dcitydb.org/3dcitydb/fileadmin/downloaddata/3DCityDB\\_Documentation\\_v3.3.pdf](https://www.3dcitydb.org/3dcitydb/fileadmin/downloaddata/3DCityDB_Documentation_v3.3.pdf). Accessed 13 Nov 2018.
- Kothuri R, Godfrind A, Beinat E (2011) *Pro Oracle Spatial for Oracle Database 11g*. Apress, New York City, USA.
- Kothuri R, Ravada S, Abugov D (2002) Quadtree and R-tree Indexes in Oracle Spatial: A Comparison using GIS Data. In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2-6 June 2002, Madison, Wisconsin USA*. ACM, pp 546-557.
- Krämer M, Gutbell R (2015) A case study on 3D geospatial applications in the web using state-of-the-art WebGL frameworks. In: *Proceedings of the 20th International Conference on 3D Web Technology, 18-21 June 2015, Heraklion, Crete, Greece*. ACM, pp 189-197.

- Krüger A, Kolbe TH (2010) A Framework for the Data-Driven Analysis, Interpretation, and Transformation of Geospatial Information Models. In: Kohlhofer G, Franzen M (eds) Proceedings of the Joint Scientific Conference of the German Society for Photogrammetry, Remote Sensing & Geoinformation (DGPF), the Austrian Society for Surveying (OVG), and the Swiss Society for Photogrammetry and Remote Sensing (SGPF), 1-3 July 2010, Vienna, Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., pp 309-324.
- Kumar K, Ledoux H, Stoter J (2016) A CityGML extension for handling very large TINs. In: Dimopoulou E, van Oosterom PJM, Zlatanova S, Isikdag U, Sithole G (eds) Proceedings of 11th 3D Geoinfo Conference, 20-21 October 2016, Athens, Greece. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-2/W1. ISPRS, pp 137-143.
- Kuske S, Gogolla M, Kreowski HJ, Ziemann P (2009) Towards an integrated graph-based semantics for UML. *Software & Systems Modeling*. 8(3):403-422.
- Kutzner T (2016) Geospatial Data Modelling and Model-driven Transformation of Geospatial Data based on UML Profiles. Dissertation, Technische Universität München.
- Kutzner T, Kolbe TH (2016) Extending Semantic 3D City Models by Supply and Disposal Networks for Analysing the Urban Supply Situation. In: Kersten TP (ed) Lösungen für eine Welt im Wandel, Dreiländertagung der SGPF, DGPF und OVG, 36. Wissenschaftlich-Technische Jahrestagung der DGPF, 7-9 Juni 2016, Bern. Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. (DGPF), vol 25, pp 382-394.
- Li Y, Li J, Zhou S (2004) GML Storage: A Spatial Database Approach. In: Wang S, Tanaka K, Zhou S, Ling TW, Guan J, Yang D, Grandi F, Mangina EE, Song IY, Mayr HC (eds) Conceptual Modelling for Advanced Application Domains. Proceedings of ER 2004 Workshops CoMoGIS, CoM-WIM, ECDM, CoMoA, DGOV, and eCOMO, 8-12 November 2004, Shanghai, China. Lecture Notes in Computer Science, vol 3289. Springer, pp 55-66.
- Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: A survey. *European Journal of Operational Research*. 141(2):241-252.
- Lodi A, Martello S, Vigo D (1999) Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. *INFORMS Journal on Computing*. 11(4):345-357.
- Loesch B, Christen M, Nebiker S (2012) OpenWebGlobe - An open source SDK for creating large-scale Virtual Globes on a WebGL basis. In: Shortis M, Madden M (eds) Proceedings of XXII ISPRS Congress, 25 August – 01 September 2012, Melbourne, Australia. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol XXXIX-B4. ISPRS, pp 195-200.
- Löwner MO, Benner J, Gröger G (2014) Aktuelle trends in der Entwicklung von CityGML 3.0. In: Seyfert E, Gülch E, Heipke C, Schiewe J, Sester M (eds) Geoinformationen öffnen das Tor zur Welt, 34. Wissenschaftlich-Technische Jahrestagung der DGPF, Hamburg. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung, Geoinformation e.V. (DGPF), vol 23.
- Maffini G (1987) Raster versus Vector Data Encoding and Handling: A Commentary. *Photogrammetric Engineering and Remote Sensing*. 53(10):1397-1398.
- Mao B, Ban Y, Harrie L (2011) A Multiple Representation Data Structure for Dynamic Visualisation of Generalised 3D City Models. *ISPRS Journal of Photogrammetry and Remote Sensing*. 66(2):198-208.
- Mao B, Harrie L, Cao J, Wu Z, Shen J (2014) NoSQL based 3D City Model Management System. In: Jiang J, Zhang H (eds) Proceedings of ISPRS Technical Commission IV Symposium, 14-16 May

- 2014, Suzhou, China. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol XL-4. ISPRS, pp 169-173.
- Marx C, Donaubaue A, Fiutak G, Kolbe TH (2017) Digitales Landschaftsmodell in 3D. In: Kolbe TH, Bill R, Donaubaue A (eds) Geoinformationssysteme 2017 - Beiträge zur 4. Münchner GI-Runde. Wichmann Verlag, Heidelberg.
- McHenry K, Bajcsy P (2008) An Overview of 3D Data Content, File Formats and Viewers. <https://www.archives.gov/files/applied-research/ncsa/8-an-overview-of-3d-data-content-file-formats-and-viewers.pdf>. Accessed 13 Nov 2018.
- Mens T, Van Gorp P (2006) A Taxonomy of Model Transformation. In: Karsai G, Taentzer G (eds) Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), 28-28 September 2005, Tallinn, Estonia. Electronic Notes in Theoretical Computer Science, vol 152. Elsevier, pp 125-142.
- Murthy R, Krishnaprasad M, Chandrasekar S, Sedlar E, Krishnamurthy V, Agarwal N (2006) Mechanism for mapping XML schemas to object-relational database systems. US Patent 7096224, 22 August 2016.
- Ng TCT, Learmont TR (2002) Rule-based approach to object-relational mapping strategies. US Patent 6360223, 19 March 2002.
- Nguyen SH, Kolbe TH (2017) Spatio-semantic Comparison of 3D City Models in CityGML using a Graph Database. In: Kalantari M, Rajabifard A (eds) Proceedings of the 12th International 3D GeoInfo Conference, 26–27 October 2017, Melbourne, Australia. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-4/W5. ISPRS, pp 99-106.
- Oracle (2017) Database Application Developer's Guide - Workspace Manager. [http://docs.oracle.com/cd/B19306\\_01/appdev.102/b14253/long\\_intro.htm](http://docs.oracle.com/cd/B19306_01/appdev.102/b14253/long_intro.htm). Accessed 13 Nov 2018.
- Ordonez C, Song IY, Garcia-Alvarado C (2010) Relational versus Non-Relational Database Systems for Data Warehousing. In: Proceedings of the ACM 13th international workshop on Data warehousing and OLAP, 26-30 October 2010, Toronto, ON, Canada. ACM, pp 67-68.
- Parisi T (2012) WebGL: Up and Running. O'Reilly Media, Inc., Sebastopol, California, United States.
- Rafe V, Jamali S, Rahmani M, Mahdian F (2011) From Class Diagrams to Relational Tables: A Graph Transformation-based Approach. *Przełąd Elektrotechniczny*. 87(8):163-165.
- Rode M, Rosenberg O (1987) An analysis of heuristic trim-loss algorithms. *Engineering Costs and Production Economics*. 12(1-4):71-78.
- Schilling A, Bolling J, Nagel C (2016) Using glTF for streaming CityGML 3D City Models. In: Proceedings of the 21st International Conference on Web3D Technology, 22-24 July 2016, Anaheim, California. ACM, pp. 109-116.
- Sencha (2017) With Ext JS, create data-intensive HTML5 applications using JavaScript. <https://www.sencha.com/products/extjs>. Accessed 13 Nov 2018.
- ShapeChange (2017) SQL DDL - ShapeChange. <http://shapechange.net/targets/sql-ddl/>. Accessed 13 Nov 2018.
- Shen Z, Kawakami M (2010) An online visualization tool for Internet-based local townscape design. *Computers Environment and Urban Systems*. 34(2):104-116.
- Shumilov S, Thomsen A, Cremers AB, Koos B (2002) Management and Visualization of large, complex and. time-dependent 3D Objects in Distributed GIS. In: Proceedings of the 10th ACM Interna-

- tional Symposium on Advances in Geographic Information Systems, 8-9 November 2002, McLean, VA (near Washington, DC), USA. ACM, pp 113-118.
- Sindram M, Machl T, Steuer H, Pültz M, Kolbe TH (2016) Voluminator 2.0 - Speeding up the Approximation of the Volume of Defective 3D Building Models. In: Halounova L, Li S, Šafář V, Tomková M, Rapant P, Brázdil K, Shi W, Anton F, Liu Y, Stein A, Cheng T, Pettit C, Li QQ, Sester M, Mostafavi MA, Madden M, Tong X, Brovelli MA, HaeKyong K, Kavashima H, Coltekin A (eds) Proceedings of XXIII ISPRS Congress, 12–19 July 2016, Prague, Czech Republic. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol III-2. ISPRS, pp 29-36.
- Siwik L, Lewandowski K, Woś A, Dreżewski R, Kisiel-Dorohinicki M (2010) UML2SQL - A Tool for Model-Driven Development of Data Access Layer. In: Szczerbicki E, Nguyen NT (eds) Smart Information and Knowledge Management. Studies in Computational Intelligence, vol 260. Springer Berlin Heidelberg, pp. 227-246.
- Software S (2016) GO Loader Documentation - XML, GML and Application Schemas. <https://wiki.snowflakesoftware.com/display/GLD/XML%2C+GML+and+Application+Schemas>. Accessed 13 Nov 2018.
- Sousa L, Eykamp C, Leopold U, Baume O, Braun C (2012) iGUESS - A web based system integrating Urban Energy Planning and Assessment Modelling for multi-scale spatial decision making. In: Seppelt R, Voinov AA, Lange S, Bankamp D (eds) Proceedings of 6th International Congress on Environmental Modelling and Software (iEMSs), 1-5 July 2012, Leipzig, Germany, pp 171-178.
- Stadler A, Kolbe TH (2007) Spatio-semantic coherence in the integration of 3D city models. In: Stein A (ed) Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality (ISSDQ), 13-15 June 2007, Enschede, Netherlands. ISPRS Archives, vol XXXVI-2/C43.
- Stadler A, Nagel C, König G, Kolbe TH (2009) Making interoperability persistent: A 3D geo database based on CityGML. In: Lee J, Zlatanova S (eds). 3D Geo-Information Sciences. Proceedings of the 3rd International Workshop on 3D Geo-Information, 13-14 November 2008, Seoul, Korea. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, pp 175-192.
- Sparx Systems (2015) Model Transformation with Enterprise Architect [http://www.sparxsystems.com/enterprise\\_architect\\_user\\_guide/9.2/model\\_transformation/mdastyle\\_transforms.html](http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/model_transformation/mdastyle_transforms.html). Accessed 13 Nov 2018.
- Taentzer G (2000) AGG: A Tool Environment for Algebraic Graph Transformation. In: Nagl M, Schürr A, Münch M (eds) Applications of Graph Transformations with Industrial Relevance. Proceedings of International Workshop, 1-3 September 1999, Kerkrade, Netherlands. Lecture Notes in Computer Science, vol 1779. Springer, Berlin, Heidelberg, pp 481-488.
- Taentzer G, Carughi GT (2006) A Graph-Based Approach to Transform XML Documents. In: Baresi L, Heckel R (eds) Fundamental Approaches to Software Engineering. Proceedings of 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, 27-28 March 2006, Vienna, Austria. Lecture Notes in Computer Science, vol 3922. Springer Berlin Heidelberg, pp 48-62.
- Taentzer G, Ehrig K, Guerra E, de Lara J, Lengyel L, Levendovszky T, Prange U, Varró D, Varró-Gyapay S (2005) Model Transformation by Graph Transformation: A Comparative Study. In: Proceedings of the workshop “Model Transformation in Practice” of the international conference on Satellite Events at the MoDELS, 2-7 October 2005, Montego Bay, Jamaica.

- Tauro CJM, Aravindh S, Shreeharsha A (2012) Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. *International Journal of Computer Applications*. 48(20):1-4.
- Thompson R, van Oosterom PJM, Cemellini B, de Vries M (2018) Developing an LADM Compliant Dissemination and Visualization System for 3D Spatial Units. In: *Proceedings of the 7th Land Administration Domain Model Workshop*, 12-13 April 2018, Zagreb, Croatia.
- OSGeo (2017) Tile Map Service Specification. [http://wiki.osgeo.org/wiki/Tile\\_Map\\_Service\\_Specification](http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification). Accessed 13 Nov 2018.
- Travers C (2012) O/R Modelling Part 1: Intro to PostgreSQL as Object-Relational Database Management System. <http://ledgersmbdev.blogspot.de/2012/08/intro-to-postgresql-as-object.html>. Accessed 13 Nov 2018.
- van Oosterom PJM, Maessen B, Quak W (2002) Generic query tool for spatio-temporal data. *International Journal of Geographical Information Science*. 16(8):713-748.
- van Oosterom PJM, Vijlbrief T (1994) Integrating Complex Spatial Analysis Functions in an Extensible GIS. In: *Proceedings of the 6th International Symposium on Spatial Data Handling*, 5-9 September 1994, Edinburgh, Scotland, pp 277-296.
- van den Brink L, Stoter J, Zlatanova S (2013) UML-Based Approach to Developing a CityGML Application Domain Extension. *Transactions in GIS*. 17(6):920-942.
- van den Broek PM (1991) Algebraic Graph Rewriting Using a Single Pushout. In: Abramsky S, Maibaum TSE (eds). *TAPSOFT '91: Proceedings of the International Joint Conference on Theory and Practice of Software Development*, 8-12 April 1991, Brighton, UK. *Lecture Notes in Computer Science*, vol 1: Colloquium on Trees in Algebra and Programming (CAAP '91). Springer Berlin Heidelberg, pp 90-102.
- Vara JM, Vela B, Bollati VA, Marcos E (2009) Supporting Model-Driven Development of Object-Relational Database Schemas: A Case Study. In: Paige RF (ed) *Theory and Practice of Model Transformations. proceedings of the Second International Conference on Theory and Practice of Model Transformations (ICMT 2009)*, 29-30 June 2009, Zurich, Switzerland. Springer Berlin Heidelberg, pp 181-196.
- Vicknair C, Macias M, Zhao Z, Nan X, Chen Y, Wilkins D (2010) A Comparison of a Graph Database and a Relational Database A Data Provenance Perspective. In: *Proceedings of the 48th Annual Southeast Regional Conference*, 15-17 April 2010, Oxford, Mississippi, USA. ACM, pp 42:1-42:6.
- Vijlbrief T, van Oosterom PJM (1992) GEO++: An extensible GIS. In: *proceedings 5th International Symposium on Spatial Data Handling*, 3-7 August 1992, Charleston, South Carolina, Humanities and Social Sciences Computing Lab, University of South Carolina, Columbia, S.C, pp 40-50.
- Faulkner S, Eicholz A, Leithead T, Danilo A, Moon S (2017) HTML 5.2 - W3C Proposed Recommendation. <http://www.w3.org/TR/html5/>. Accessed 13 Nov 2018.
- WebGL Earth (2017) WebGL Earth – open source 3D digital globe written in JavaScript. <http://www.webglearth.org/>. Accessed 13 Nov 2018.
- Web3D (2015) X3D Standards for Version v3.3. <https://www.web3d.org/standards/version/V3.3>. Accessed 13 Nov 2018.
- Westra E (2010) *Python Geospatial Development*. Packt Publishing Ltd., Birmingham, UK.



- Wloka M (2005) Improved Batching via Texture Atlases. In: Engel W (ed) Shader X3: Advanced Rendering with DirectX and OpenGL. Charles River Media, Newton Center, MA, USA, pp 155-167.
- Wu H, He Z, Gong J (2010) A virtual globe-based 3D visualization and interactive framework for public participation in urban planning processes. *Computers, Environment and Urban Systems*. 34(4):291-298.
- Yao Z, Chaturvedi K, Kolbe TH (2016) Browser-basierte Visualisierung großer 3D-Stadtmodelle durch Erweiterung des Cesium Web Globe. In: Kolbe TH, Bill R, Donaubaue A (eds) *Geoinformationssysteme 2016 - Beiträge zur 3. Münchner GI-Runde, 24-25 February 2016*. Wichmann Verlag Heidelberg.
- Yao Z, Kolbe TH (2017) Dynamically Extending Spatial Databases to support CityGML Application Domain Extensions using Graph Transformations. In: Kersten TP (ed) *Kulturelles Erbe erfassen und bewahren - Von der Dokumentation zum virtuellen Rundgang, 37. Wissenschaftlich-Technische Jahrestagung der DGPF, 7-10 March 2016, Würzburg*. Publikationen von Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. (DGPF), vol 26. pp. 316-331.
- Yao Z, Sindram M, Kaden R, Kolbe TH (2014) Cloud-basierter 3D-Webclient zur kollaborativen Planung energetischer Maßnahmen am Beispiel von Berlin und London. In: Kolbe TH, Bill R, Donaubaue A (eds) *Geoinformationssysteme 2014: Beiträge zur 1. Münchner GI-Runde, 24-25 February 2014*. Wichmann Verlag Heidelberg.
- Zahariev A (2009). Google App Engine.  
[http://cse.tkk.fi/en/publications/B/5/papers/1Zahariev\\_final.pdf](http://cse.tkk.fi/en/publications/B/5/papers/1Zahariev_final.pdf). Accessed 13 Nov 2018.
- Zheng Y, Capra L, Wolfson O, Yang H (2014) Urban Computing: Concepts, Methodologies, and Applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*. 5(3): 38:1-38:55
- Zulkifli NA, Rahman AA, Jamil H, Hua TC, Choon TL, Seng LK, Lim CK, van Oosterom PJM (2014) Development of a Prototype for the Assessment of the Malaysian LADM Country Profile. In: *Proceedings of the 25th FIG Congress, 16 – 21 June 2014, Kuala Lumpur, Malaysia*.

All URLs in this thesis were checked on January 31, 2019.





## Appendix 1: Layered Graph Transformation Rules

This appendix complements section 4.3.1 by providing more details about some selected graph transformation rules as well as their group affiliations for realizing the designed layered graph transformation.

Rule 1: Creating singleton 3DCityDB tables	
Group Layer	1
Description	<p>As mentioned in section 4.3.1, the first transformation layer is designed for mapping each feature class or complex data type onto an individual table. Besides, this layer also includes a few additional transformation rules for initializing three singleton graph representations in the host graph for representing the three predefined 3DCityDB tables, namely OBJECTCLASS, SURFACE_GEOMETRY, and IMPLICIT_GEOMETRY, which are intended to be utilized for storing the meta und spatial information of ADE database schemas. In order to represent each of these three 3DCityDB tables in the host graph, only two associated nodes are required which represent the table and its primary key respectively and allow for building foreign key connections with ADE tables. For creating the graph nodes, the following graph transformation rules have been designed, where the LHS is blank and RHS contains the table node and its respective primary key node. Each rule is additionally equipped with a NAC which is morphologically identical with the RHS to ensure that this transformation will be performed only one time for ensuring the creation of three singleton table nodes.</p>
Graph Definition of the Transformation rule A	
Graph Definition of the Transformation rule B	
Graph Definition of the Transformation rule C	

<b>Rule 2: Mapping multiple classes onto one table</b>	
Group Layer	2
Description	<p>The second transformation layer holds those graph transformation rules that tend to merge multiple classes onto one table which is mapped from their super class. Thus, this layer plays an important role for creating a compact relational model, as the number of tables can be reduced. In the section 4.3.1, an example of using the graph transformation rule of this layer is outlined which allows for handling the classes with <i>Composite Pattern</i> to derive an optimized relational database structure. Generally, depending on the various user-defined conditions or application specific requirements, this model optimization process can be realized in a generic way. This can be easily done by modifying the definition of the NACs of the following graph transformation rule. For example, if a subclass must be mapped onto an own table, if it has more than four properties, then we can define a NAC where a class node is connected with four nodes typed as <code>propertyElement</code>. This way, the graph transformation rule for merging subclasses is only applicable when the subclass node is connected with maximum three property nodes.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

<b>Rule 3: Mapping class inheritances onto foreign key constraints</b>	
Group Layer	3
Description	<p>The third transformation layer is mainly designed for creating foreign key constraint for each inheritance relationship from two classes, which are not mapped onto the same table. In the graph transformation rule, a new graph node typed as <code>Join</code> is created for representing the foreign key constraint which connects the primary key columns of the two tables mapped from the super class and subclass respectively. The foreign key constraint is named by following the simple naming convention:</p> $[FK\_Name] = M + \_FK$ <p>Where the variable “M” denotes the name of the super table. In order to avoid creating the foreign key for each pair of classes twice, a NAC is added to the graph transformation rule. In the NAC, the graph node representing the inheritance relationship has a mapping onto a graph node representing the foreign key constraint.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 4: Creating ObjectClassID column for merged tables (1)	
Group Layer	3
Description	<p>The third graph transformation layer also includes some additional rules for adding an OBJECTCLASS_ID column to each of those tables that are mapped from multiple non-abstract classes. The OBJECTCLASS_ID is a foreign key column referencing to the primary key of the OBJECTCLASS table is required for providing the affiliation information, as the table rows may store attribute information of different feature class instances. The logical conditions of this mapping rule are summarized as the following:</p> <ul style="list-style-type: none"> <li>• The graph node representing the OBJECTCLASS table has already created</li> <li>• At least two non-abstract classes have been already mapped onto the same table</li> </ul> <p>In the graph transformation rule, a node typed as <code>ObjectclassIdColumn</code> is used for representing the OBJECTCLASS_ID column and a node with the type <code>Join</code> serves for representing the corresponding foreign key constraint referencing to the the OBJECTCLASS table.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

<b>Rule 5: Creating ObjectClassID column for merged tables (2)</b>	
Layer group	3
Description	<p>In case that a class is mapped onto one table and its subclasses are mapped onto individual tables, the parent table may also contain rows storing attribute information coming from different classes and shall be hence augmented with an OBJECTCLASS_ID column. the logical mapping of this scenario and the corresponding graph transformation rule are shown below. Compared with the LHS, two graph nodes representing the OBJECTCLASS_ID column and its foreign key constraint are created in the RHS and connected with the parent class table and OBJECTCLASS table. To ensure that only one OBJECTCLASS_ID column will be created for the target parent table, a NAC is required for suppressing the execution of this graph transformation rule again when the OBJECTCLASS_ID column has already been created and connected with the target class table in the host graph.</p>
Logical Mapping	
Graph Definition of the Transformation rule	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>LHS</p> </div> <div style="width: 48%;"> <p>RHS</p> </div> </div>
Negative Application Condition (NAC)	

<b>Rule 6: Creating ObjectClassID column for merged tables (3)</b>	
Layer group	3
Description	<p>If a class is mapped onto a table having an OBJECTCLASS_ID column and the super class is mapped onto a separate table, then the super class table shall also be equipped with an OBJECTCLASS_ID column. This is because that the Polymorphism can be transitively propagated from the subclass to its super class which can hence also contain attribute information coming from different classes. The following figure shows the logical structure of this mapping scenario and the corresponding graph transformation rule has been implemented as shown in the figure below. similar with the previous case, two new nodes representing the OBJECTCLASS_ID column and its corresponding foreign key constraint are created in the RHS and a NAC is also added to the graph transformation rule for guaranteeing that only one OBJECTCLASS_ID column will be created for each table.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	



<b>Rule 7: Mapping N:0..1 Association between different classes</b>	
Layer group	4
Description	<p>Starting from the third transformation layer, the class associations are mapped onto foreign key constraints or associative tables depending on the multiplicity of each association. These kinds of mapping rules are explained in the section 3.2.1. The first case is the mapping of two different classes with a N:0..1 association, which is logically mapped onto a foreign key constraint. In the graph transformation rule, the two graph nodes with the type <code>ComplexTypeProperty</code> has a numeric attribute <code>maxOccurs</code> which defines the upper bound of the multiplicity. Using this attribute, an application condition <math>M == 1</math> can be added to the graph transformation rule in order to trigger the corresponding graph transformation rule. In the RHS, two new nodes representing the foreign key column and constraints are created to connect the associated tables. Their names can be defined according to the following rule:</p> $[FK\_Name] = Y + \_ + n + \_ \_ FK$ $[FK\_COLUMN\_NAME] = n + \_ \_ ID$ <p>Where Y denotes the referencing table name, and N denotes the property name.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 8: Mapping N:0..1 Association between different classes mapped to the same table	
Layer group	4
Description	<p>Another case of the N:0..1 association is that the associated classes are mapped onto the same table. Also, the association is logically mapped onto a foreign key constraint for which the referencing and referenced tables are identical. Similar to the previous graph transformation rule, the numeric attribute <code>maxOccurs</code> of the graph node <code>ComplexTypeProperty</code> has been used for defining the application condition <math>M == 1</math>, which means that the transformation rule is only applicable when the upper bound of the association multiplicity is 1. The naming rules for the newly created graph nodes in the RHS is defined as the following:</p> $[FK\_Name] = X + \_ + N + \_FK$ $[FK\_COLUMN\_NAME] = N + \_ID$ <p>Where <math>X</math> denotes the mapped table name, and <math>N</math> denotes the role name of the corresponding association.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 9: Mapping N:0..1 Association between the same class mapped to the same table	
Layer group	4
Description	<p>The last case of the N:0..1 association is that the associated classes are identical and mapped onto the same table. The logical mapping and graph transformation rule are very similar to those of the previous case and shown in the figures below. The naming rules for the created foreign key column and constraint are also adopted from the previous case and are formalized as below;</p> $[FK\_Name] = X + \_ + N + \_FK$ $[FK\_COLUMN\_NAME] = N + \_ID$ <p>Where X denotes the referencing table name, and N denotes the role name of the corresponding association.</p> <p>As explained in the section 3.2.1, the 1:N association can also be categorized into three different scenarios compared to the N:0..1 association and their graph transformation rules are also structured in a very similar way. For the sake of brevity, these rules are not outlined in this section and the details of technical implementation can be found in on the 3DCityDB Github website.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 10: Mapping M:N association between different classes	
Layer group	4
Description	<p>For M:N association an associative table is required that includes two foreign key columns to reference to the primary key columns of the associated tables. According to the logical mapping shown below, two graph nodes with the type <code>ComplexTypeProperty</code> are used for representing the binary association in the host graph and each has a numeric attribute <code>maxOccurs</code> to define the upper bound of the multiplicity value:</p> $(M > 1 \    \ M == -1) \ \&\& \ (N > 1 \    \ N > -1)$ <p>where M and N represent the <code>maxOccurs</code> attribute of the two associated classes and the value of -1 indicates that the multiplicity is unbounded. The associative table can be named based on the following rule:</p> $[tableName] = A + \_ + B + \_TO\_ + C + \_ + D$ <p>Where A and C are the names of associated tables, while B and D denotes the role names of the two association ends.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 11: Mapping M:N among the same class	
Layer group	4
Description	<p>two special cases of the M:N association are that, 1): a class is associated with itself and 2): two associated classes are mapped onto the same table. Both cases have a similar graph representation in the graph transformation rule for representing the derived associative table in the RHS. Again, the numeric attribute <code>maxOccurs</code> of the two <code>ComplexTypeProperty</code> nodes are used for defining the application condition:</p> $(M > 1 \    \ M == -1) \ \&\& \ (N > 1 \    \ N > -1)$ <p>where <code>M</code> and <code>N</code> represent the upper bound of the multiplicity. The associative table is can be named according to a simplified rule:</p> $[tableName] = A + \_ + B + \_ + C$ <p>where <code>A</code>, <code>B</code>, and <code>C</code> represent the names of class table and role names of the two association ends respectively.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 12: Mapping non-Brep geometry property	
Layer group	5
Description	<p>In the fifth transformation layer, every thematic and spatial property of each class and complex data type will be mapped onto one or multiple columns or a foreign key constraint referencing to a separate data table which stores the property value. The latter case is typically used for the handling of the Brep-geometry property using the SURFACE_GEOMETRY table and has already been mentioned in the section 4.3.1. For the non-Brep geometry property, it can be simply mapped onto a column with a spatial type (e.g. for Oracle, the spatial type is called "SDO_GEOMETRY") in the class table and the column name can be identical with the geometry property name. The graph implementation of this logical mapping rule is quite simple: A new graph node typed with <code>InlineGeometryColumn</code> is created in the RHS to connect with the class table node and an additional NAC is defined for avoiding the creation of the geometry column twice.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

Rule 13: Mapping hybrid geometry property	
Layer group	5
Description	<p>In practice, a feature class or complex data type may have a generic geometry property which allows to have a Brep or non-Brep representation at the same time. This kind of geometry properties shall hence be mapped onto a spatial column in the class table along with a foreign key column referencing to the SURFACE_GEOMETRY column. This way, all non-Brep-geometry contents are stored in the spatial column and the storage of Brep-geometry data are delegated to the SURFACE_GEOMETRY table. The logical mapping relationship between the object-oriented model and relational database model is shown below. The implementation of the corresponding graph transformation rule is somehow like the combination of the rules for handling Brep and non-Brep geometry properties. Also, a NAC is added for avoiding the creation of the geometry column two times.</p>
Logical Mapping	<p>The diagram illustrates the logical mapping between an Object-Oriented Model (OOM) and a Relational Database Model (RDBM). In the OOM, a «FeatureType» class (labeled «ClassName») contains a «HybridGeometry» object (labeled «ObjectType»). The relationship is shown as 0..1. In the RDBM, the «FeatureType» is mapped to a table «&lt;TableNameForClass1&gt;» with columns: «PK ID: NUMBER» (primary key), «FK BREP_ID: NUMBER» (foreign key), and «OTHER_GEOM: SDO_GEOMETRY». A foreign key relationship «FK» is established between «FK BREP_ID» and a primary key «(BREP_ID = ID)» in the «SURFACE_GEOMETRY» table, which has a column «pk ID: NUMBER».</p>
Graph Definition of the Transformation rule	<p>The graph definition is split into LHS (Left Hand Side) and RHS (Right Hand Side).  <b>LHS:</b> Node 1:ComplexType mapsTo (6) node 2:DataTable and contains (7) node 3:HybridGeometryProperty. Node 3:HybridGeometryProperty belongsTo (8) node 4:PrimaryKeyColumn. Node 4:PrimaryKeyColumn belongsTo (8) node 5:DataTable.  <b>RHS:</b> Node 1:ComplexType contains (7) node 3:HybridGeometryProperty. Node 3:HybridGeometryProperty mapsTo (6) node 2:DataTable and mapsTo (6) node 4:RefGeometryColumn. Node 3:HybridGeometryProperty mapsTo (6) node 4:RefGeometryColumn. Node 4:RefGeometryColumn belongsTo (8) node 2:DataTable. Node 4:RefGeometryColumn belongsTo (8) node 5:DataTable. Node 4:RefGeometryColumn joinFrom node 4:RefGeometryColumn. Node 4:RefGeometryColumn joinTo node 4:RefGeometryColumn. Node 4:RefGeometryColumn joinTo node 4:RefGeometryColumn.</p>
Negative Application Condition	<p>The Negative Application Condition (NAC) graph shows a central node 3:HybridGeometryProperty with two outgoing mapsTo relationships to nodes InlineGeometryColumn and RefGeometryColumn.</p>



Rule 14: Mapping Implicit Geometry	
Layer group	5
Description	<p>Another important geometry type of the CityGML's spatial model is the <i>Implicit Geometry</i> which is based on the scene graph concept and can also be used in CityGML ADEs (cf. Gröger et al. 2012). The relational representation of the implicit geometry property has already been implemented in the standard 3DCityDB database schema based on which a foreign key column <code>LOD[X]_IMPLICIT_REF_ID</code> is firstly created which references to the primary key of the <code>IMPLICIT_GEOMETRY</code> table. Further, two additional columns <code>LOD[X]_IMPLICIT_REF_POINT</code> and <code>LOD[X]_IMPLICIT_TRANSFORMATION</code> shall be added for providing the information on how to place the implicit geometry in the real-world coordinate reference systems. Note that the token <code>[x]</code> denotes the level of detail (LOD) of the respective implicit geometry representation. The corresponding implemented graph transformation rule are shown in the figures below.</p>
Logical Mapping	
Graph Definition of the Transformation rule	
Negative Application Condition	

<b>Rule 15: Creating Index for foreign key and spatial columns</b>	
Layer group	6
Description	<p>The second-last graph transformation layer contains only two simple graph transformation rules for creating database indexes for the foreign key columns and spatial columns. It is usually necessary to create indexes on foreign key columns for avoiding the full table scan when updating the table contents in order to achieve a better database performance. Furthermore, the foreign key index is especially important for the database like Oracle, because the 'ON DELETE CASCADE' operation may cause a "table-level" lock issue while performing the delete operations on the tables, when no indexes are created on the corresponding foreign key columns. For spatial columns, the spatial indexes are important not only for guaranteeing good query performance but also required e.g. by Oracle for its spatial operators like SDO_FILTER, SDO_ANYINTERACT, and SDO_JOIN etc. (cf. Kothuri et al. 2011)</p> <p>To create these two kinds of indexes, the following two graph transformation rules are designed in this graph transformation layer. Compared with the LHS of each graph transformation rule, a new graph node representing the database index is added into the RHS and an additional edge typed as <code>targetColumn</code> is created to connect the index node with its respective column. Since only one index shall be created on the corresponding column, each graph transformation rule also has a NAC which is morphologically identical with the respective RHS and allows to ensure that the transformation rule will be applied on the target column only one time.</p>
Graph Definition of the Transformation rule A	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>LHS</p> <pre> graph TD     2:Join["2:Join name=n"] -- 3:joinFrom --&gt; 1:JoinColumn["1:JoinColumn"]                     </pre> </div> <div style="width: 45%;"> <p>RHS</p> <pre> graph TD     2:Join["2:Join name=n"] -- 3:joinFrom --&gt; 1:JoinColumn["1:JoinColumn"]     1:JoinColumn -- targetColumn --&gt; Index["Index name=n+'X' isSpatialIndex=false"]                     </pre> </div> </div>
Graph Definition of the Transformation rule B	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>LHS</p> <pre> graph TD     2:DataTable["2:DataTable name=m"] -- 3:belongsTo --&gt; 1:InlineGeometryColumn["1:InlineGeometryColumn name=n"]                     </pre> </div> <div style="width: 45%;"> <p>RHS</p> <pre> graph TD     2:DataTable["2:DataTable name=m"] -- 3:belongsTo --&gt; 1:InlineGeometryColumn["1:InlineGeometryColumn name=n"]     1:InlineGeometryColumn -- targetColumn --&gt; Index["Index name=m+'_'+'n'+'_SPX' isSpatialIndex=true"]                     </pre> </div> </div>

<b>Rule 16: Creating Sequence for tables mapped from non-Feature(Object) classes</b>	
Layer group	7
Description	<p>The last transformation layer contains only one simple transformation rule for creating a database sequence for each of those tables that are mapped from the classes modelled with the stereotypes &lt;&lt;DataType&gt;&gt; or &lt;&lt;Union&gt;&gt;. The database sequence allows to automatically generate incremental integer values which are hence unique and can be used as primary key values for the respective table. Note that it is not necessary to create sequence for the tables mapped from &lt;&lt;FeatureType&gt;&gt; and &lt;&lt;ObjectType&gt;&gt; classes, because all this kind of tables are directly or transitively subtyped from the 3DCityDB table CITYOBJECT for which a sequence CITYOBJECT_SEQ has already been provided in the standard 3DCityDB database schema.</p> <p>The graph transformation rule is shown below, where a new node typed as Sequence is created in the RHS and connected with the target table via a graph edge typed as targetTable. The name of the newly created sequence is defined by following the naming convention used for the 3DCityDB:</p> <p style="text-align: center;">[Sequecne Name] = [Target Table Name] + "_SEQ"</p> <p>This transformation rule is additionally augmented with the following application condition</p> <p style="text-align: center;">x != "_Feature" &amp;&amp; x != "_GML" &amp;&amp; b = false</p> <p>which means that the complex type should not be a &lt;&lt;FeatureType&gt;&gt; or &lt;&lt;ObjectType&gt;&gt; and it should belong to the scope of the target ADE, not to GML or CityGML. Similar with the other graph transformation rules, a NAC is defeind for ensuring that only one sequence object will be created for the target table.</p>
Graph Definition of the Transformation rule A	
Negative Application Conditions	

## Appendix 2: XML Schema Definition File of the TestADE

The following XML schema definition of the TestADE (cf. chapter 4.3.3) has been automatically generated from its EA-UML model using the software tool ShapeChange (cf. chapter 4.3.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:TestADE="http://www.citygml.org/ade/TestADE/1.0"
xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
xmlns:core="http://www.opengis.net/citygml/2.0"
xmlns:gml="http://www.opengis.net/gml" xmlns:sc="http://www.interactive-
instruments.de/ShapeChange/AppInfo"
elementFormDefault="qualified" target-
Namespace="http://www.citygml.org/ade/TestADE/1.0"
version="1.0">
  <import namespace="http://www.interactive-instruments.de/ShapeChange/AppInfo"
    schemaLocation="http://shapechange.net/resources/schema/ShapeChangeAppinfo.xsd"
  />
  <import namespace="http://www.opengis.net/citygml/2.0"
    schemaLocation="http://schemas.opengis.net/citygml/2.0/cityGMLBase.xsd" />
  <import namespace="http://www.opengis.net/citygml/building/2.0"
    schemaLocation="http://schemas.opengis.net/citygml/building/2.0/building.xsd"
  />
  <import namespace="http://www.opengis.net/gml"
    schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" />
  <!--XML Schema document created by ShapeChange - http://shapechange.net/ -->
  <element name="ownerName"
    substitutionGroup="bldg:_GenericApplicationPropertyOfAbstractBuilding"
    type="string" />
  <element name="floorArea"
    substitutionGroup="bldg:_GenericApplicationPropertyOfAbstractBuilding"
    type="gml:AreaType" />
  <element name="energyPerformanceCertification"
    substitutionGroup="bldg:_GenericApplicationPropertyOfAbstractBuilding"
    type="TestADE:EnergyPerformanceCertificationPropertyType" />
  <element name="buildingUnit"
    substitutionGroup="bldg:_GenericApplicationPropertyOfAbstractBuilding"
    type="TestADE:_AbstractBuildingUnitPropertyType" />
  <element name="BuildingUnit" substitutionGroup="TestADE:_AbstractBuildingUnit"
    type="TestADE:BuildingUnitType" />
  <complexType name="BuildingUnitType">
    <complexContent>
      <extension base="TestADE:_AbstractBuildingUnitType">
        <sequence />
      </extension>
    </complexContent>
  </complexType>
  <complexType name="BuildingUnitPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:BuildingUnit" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="BuildingUnitPart" substitu-
tionGroup="TestADE:_AbstractBuildingUnit"
    type="TestADE:BuildingUnitPartType" />
  <complexType name="BuildingUnitPartType">
    <complexContent>
      <extension base="TestADE:_AbstractBuildingUnitType">
        <sequence />
      </extension>
    </complexContent>
  </complexType>
  <complexType name="BuildingUnitPartPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:BuildingUnitPart" />
    </sequence>
  </complexType>
```

```

    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="DHWFacilities" substitutionGroup="TestADE:Facilities"
    type="TestADE:DHWFacilitiesType" />
  <complexType name="DHWFacilitiesType">
    <complexContent>
      <extension base="TestADE:FacilitiesType">
        <sequence />
      </extension>
    </complexContent>
  </complexType>
  <complexType name="DHWFacilitiesPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:DHWFacilities" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="EnergyPerformanceCertification"
    substitutionGroup="gml:_Object"
    type="TestADE:EnergyPerformanceCertificationType" />
  <complexType name="EnergyPerformanceCertificationType">
    <sequence>
      <element maxOccurs="unbounded" name="certificationName"
        type="string" />
      <element name="certificationid" type="string" />
    </sequence>
  </complexType>
  <complexType name="EnergyPerformanceCertificationPropertyType">
    <sequence>
      <element ref="TestADE:EnergyPerformanceCertification" />
    </sequence>
  </complexType>
  <element abstract="true" name="Facilities" substitutionGroup="gml:_Feature"
    type="TestADE:FacilitiesType" />
  <complexType abstract="true" name="FacilitiesType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element name="totalValue" type="gml:MeasureType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="FacilitiesPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:Facilities" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="IndustrialBuilding" substitutionGroup="bldg:_AbstractBuilding"
    type="TestADE:IndustrialBuildingType">
    <annotation>
      <appinfo>
        <taggedValue
          xmlns="http://www.interactive-instruments.de/ShapeChange/AppInfo"
          tag="topLevel">true</taggedValue>
      </appinfo>
    </annotation>
  </element>
  <complexType name="IndustrialBuildingType">
    <complexContent>
      <extension base="bldg:AbstractBuildingType">
        <sequence>
          <element minOccurs="0" name="remark" type="string" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IndustrialBuildingPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:IndustrialBuilding" />
    </sequence>
  </complexType>

```

```

    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="IndustrialBuildingPart" substitutionGroup="bldg:BuildingPart"
    type="TestADE:IndustrialBuildingPartType" />
  <complexType name="IndustrialBuildingPartType">
    <complexContent>
      <extension base="bldg:BuildingPartType">
        <sequence>
          <element minOccurs="0" name="remark" type="string" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IndustrialBuildingPartPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:IndustrialBuildingPart" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="IndustrialBuildingRoofSurface"
    substitutionGroup="bldg:RoofSurface"
    type="TestADE:IndustrialBuildingRoofSurfaceType" />
  <complexType name="IndustrialBuildingRoofSurfaceType">
    <complexContent>
      <extension base="bldg:RoofSurfaceType">
        <sequence>
          <element minOccurs="0" name="remark" type="string" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IndustrialBuildingRoofSurfacePropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:IndustrialBuildingRoofSurface" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="LightingFacilities" substitutionGroup="TestADE:Facilities"
    type="TestADE:LightingFacilitiesType" />
  <complexType name="LightingFacilitiesType">
    <complexContent>
      <extension base="TestADE:FacilitiesType">
        <sequence />
      </extension>
    </complexContent>
  </complexType>
  <complexType name="LightingFacilitiesPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:LightingFacilities" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element name="OtherConstruction" substitutionGroup="core:_Site"
    type="TestADE:OtherConstructionType">
    <annotation>
      <appinfo>
        <taggedValue
          xmlns="http://www.interactive-instruments.de/ShapeChange/AppInfo"
          tag="topLevel">true</taggedValue>
      </appinfo>
    </annotation>
  </element>
  <complexType name="OtherConstructionType">
    <complexContent>
      <extension base="core:AbstractSiteType">
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" name="boundedBy"
            type="bldg:BoundarySurfacePropertyType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

```

```

    </complexContent>
  </complexType>
  <complexType name="OtherConstructionPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:OtherConstruction" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
  <element abstract="true" name="_AbstractBuildingUnit"
    substitutionGroup="core:_CityObjectType" type="TestADE:_AbstractBuildingUnitType"
  />
  <complexType abstract="true" name="_AbstractBuildingUnitType">
    <complexContent>
      <extension base="core:AbstractCityObjectType">
        <sequence>
          <element minOccurs="0" name="class" type="gml:CodeType" />
          <element maxOccurs="unbounded" minOccurs="0" name="usage"
            type="gml:CodeType" />
          <element maxOccurs="unbounded" minOccurs="0" name="function"
            type="gml:CodeType" />
          <element maxOccurs="unbounded" minOccurs="0"
            name="energyPerformanceCertification"
            type="TestADE:EnergyPerformanceCertificationPropertyType" />
          <element minOccurs="0" name="lod2MultiCurve"
            type="gml:MultiCurvePropertyType" />
          <element minOccurs="0" name="lod3MultiCurve"
            type="gml:MultiCurvePropertyType" />
          <element minOccurs="0" name="lod4MultiCurve"
            type="gml:MultiCurvePropertyType" />
          <element minOccurs="0" name="lod1MultiSurface"
            type="gml:MultiSurfacePropertyType" />
          <element minOccurs="0" name="lod2MultiSurface"
            type="gml:MultiSurfacePropertyType" />
          <element minOccurs="0" name="lod3MultiSurface"
            type="gml:MultiSurfacePropertyType" />
          <element minOccurs="0" name="lod4MultiSurface"
            type="gml:MultiSurfacePropertyType" />
          <element minOccurs="0" name="lod1Solid" type="gml:SolidPropertyType" />
          <element minOccurs="0" name="lod2Solid" type="gml:SolidPropertyType" />
          <element minOccurs="0" name="lod3Solid" type="gml:SolidPropertyType" />
          <element minOccurs="0" name="lod4Solid" type="gml:SolidPropertyType" />
          <element maxOccurs="unbounded" minOccurs="0" name="address"
            type="core:AddressPropertyType" />
          <element maxOccurs="unbounded" minOccurs="0"
            name="equippedWith" type="TestADE:FacilitiesPropertyType" />
          <element maxOccurs="unbounded" minOccurs="0" name="consistsOf"
            type="TestADE:BuildingUnitPartPropertyType" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="_AbstractBuildingUnitPropertyType">
    <sequence minOccurs="0">
      <element ref="TestADE:_AbstractBuildingUnit" />
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup" />
  </complexType>
</schema>

```



## Appendix 3: SQL Definition of the TestADE Oracle DB-Schema

The following Oracle-compliant SQL definition of the TestADE (cf. chapter 4.3.3) has been automatically generated from its XML schema definition file using the developed graph-based model transformation tool (cf. chapter 4.3.1).

```
-- This document was automatically created by the ADE-Manager tool of 3DCityDB
(https://www.3dcitydb.org) on 2017-09-20 11:50:06
--
+++++
+++++
-- ***** Create tables
-- *****
--
+++++
+++++
-----
-- test_BuildingU_to_address
-----
CREATE TABLE test_BuildingU_to_address
(
    BuildingUnit_ID INTEGER NOT NULL,
    address_ID INTEGER NOT NULL,
    PRIMARY KEY (BuildingUnit_ID, address_ID)
);
-----
-- test_BuildingUnit
-----
CREATE TABLE test_BuildingUnit
(
    ID INTEGER NOT NULL,
    OBJECTCLASS_ID INTEGER,
    BuildingUnit_Parent_ID INTEGER,
    BuildingUnit_Root_ID INTEGER,
    building_buildingUnit_ID INTEGER,
    class_codespace VARCHAR2(254),
    class VARCHAR2(254),
    usage_codespace VARCHAR2(254),
    usage VARCHAR2(254),
    function_codespace VARCHAR2(254),
    function VARCHAR2(254),
    lod2MultiCurve MDSYS.SDO_GEOMETRY,
    lod3MultiCurve MDSYS.SDO_GEOMETRY,
    lod4MultiCurve MDSYS.SDO_GEOMETRY,
    lod1MultiSurface_ID INTEGER,
    lod2MultiSurface_ID INTEGER,
    lod3MultiSurface_ID INTEGER,
    lod4MultiSurface_ID INTEGER,
    lod1Solid_ID INTEGER,
    lod2Solid_ID INTEGER,
    lod3Solid_ID INTEGER,
    lod4Solid_ID INTEGER,
    PRIMARY KEY (ID)
);
-----
-- test_EnergyPerformanceCer
-----
CREATE TABLE test_EnergyPerformanceCer
(
    ID INTEGER NOT NULL,
    BuildingUni_energyPerf_ID INTEGER,
    certificationName VARCHAR2(254),
    certificationid VARCHAR2(254),
    PRIMARY KEY (ID)
);
```

```
-----  
-- test_Facilities  
-----  
CREATE TABLE test_Facilities  
(  
  ID INTEGER NOT NULL,  
  OBJECTCLASS_ID INTEGER,  
  BuildingUni_equippedWi_ID INTEGER,  
  totalValue_uom VARCHAR2(254),  
  totalValue NUMBER,  
  PRIMARY KEY (ID)  
);  
  
-----  
-- test_IndustrialBuilding  
-----  
CREATE TABLE test_IndustrialBuilding  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR2(254),  
  PRIMARY KEY (ID)  
);  
  
-----  
-- test_IndustrialBuildingPa  
-----  
CREATE TABLE test_IndustrialBuildingPa  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR2(254),  
  PRIMARY KEY (ID)  
);  
  
-----  
-- test_IndustrialBuildingRo  
-----  
CREATE TABLE test_IndustrialBuildingRo  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR2(254),  
  PRIMARY KEY (ID)  
);  
  
-----  
-- test_OtherConstruction  
-----  
CREATE TABLE test_OtherConstruction  
(  
  ID INTEGER NOT NULL,  
  PRIMARY KEY (ID)  
);  
  
-----  
-- test_Other_to_thema_surfa  
-----  
CREATE TABLE test_Other_to_thema_surfa  
(  
  OtherConstruction_ID INTEGER NOT NULL,  
  thematic_surface_ID INTEGER NOT NULL,  
  PRIMARY KEY (OtherConstruction_ID, thematic_surface_ID)  
);  
  
-----  
-- test_building  
-----  
CREATE TABLE test_building  
(  
  ID INTEGER NOT NULL,  
  ownerName VARCHAR2(254),  
  floorArea_uom VARCHAR2(254),  
  floorArea NUMBER,
```

```

EnergyPerfor_certificatio VARCHAR2(254),
EnergyPerfo_certificati_1 VARCHAR2(254),
PRIMARY KEY (ID)
);

--
+++++
+++++
-- ***** Create foreign keys
*****
--
+++++
+++++
-----
-- test_BuildingU_to_address
-----
ALTER TABLE test_BuildingU_to_address
ADD CONSTRAINT test_Buildi_to_addres_FK1 FOREIGN KEY (BuildingUnit_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingU_to_address
ADD CONSTRAINT test_Buildi_to_addres_FK2 FOREIGN KEY (address_ID) REFERENCES
address (ID);

-----
-- test_BuildingUnit
-----
ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_Objectcl_FK FOREIGN KEY (OBJECTCLASS_ID)
REFERENCES objectclass (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUnit_FK FOREIGN KEY (ID) REFERENCES cityobject
(ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUn_Parent_FK FOREIGN KEY (BuildingUnit_Parent_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUnit_Root_FK FOREIGN KEY (BuildingUnit_Root_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Build_build_build_FK FOREIGN KEY (building_buildingUnit_ID)
REFERENCES test_building (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod1Mult_FK FOREIGN KEY (lod1MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod2Mult_FK FOREIGN KEY (lod2MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod3Mult_FK FOREIGN KEY (lod3MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod4Mult_FK FOREIGN KEY (lod4MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod1Soli_FK FOREIGN KEY (lod1Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod2Soli_FK FOREIGN KEY (lod2Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

```

```

ALTER TABLE test_BuildingUnit
  ADD CONSTRAINT test_Building_lod3Soli_FK FOREIGN KEY (lod3Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
  ADD CONSTRAINT test_Building_lod4Soli_FK FOREIGN KEY (lod4Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

-----
-- test_EnergyPerformanceCer
-----
ALTER TABLE test_EnergyPerformanceCer
  ADD CONSTRAINT test_Energ_Build_energ_FK FOREIGN KEY (BuildingU-
ni_energPerf_ID) REFERENCES test_BuildingUnit (ID);

-----
-- test_Facilities
-----
ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Faciliti_Objectcl_FK FOREIGN KEY (OBJECTCLASS_ID)
REFERENCES objectclass (ID);

ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Facilities_FK FOREIGN KEY (ID) REFERENCES cityobject (ID);

ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Facil_Build_equip_FK FOREIGN KEY (BuildingU-
ni_equippedWi_ID) REFERENCES test_BuildingUnit (ID);

-----
-- test_IndustrialBuilding
-----
ALTER TABLE test_IndustrialBuilding
  ADD CONSTRAINT test_IndustrialBuildin_FK FOREIGN KEY (ID) REFERENCES building
(ID);

-----
-- test_IndustrialBuildingPa
-----
ALTER TABLE test_IndustrialBuildingPa
  ADD CONSTRAINT test_IndustrialBuild_FK_1 FOREIGN KEY (ID) REFERENCES building
(ID);

-----
-- test_IndustrialBuildingRo
-----
ALTER TABLE test_IndustrialBuildingRo
  ADD CONSTRAINT test_IndustrialBuild_FK_2 FOREIGN KEY (ID) REFERENCES themat-
ic_surface (ID);

-----
-- test_OtherConstruction
-----
ALTER TABLE test_OtherConstruction
  ADD CONSTRAINT test_OtherConstruction_FK FOREIGN KEY (ID) REFERENCES cityobject
(ID);

-----
-- test_Other_to_thema_surfa
-----
ALTER TABLE test_Other_to_thema_surfa
  ADD CONSTRAINT test_Othe_to_them_sur_FK1 FOREIGN KEY (OtherConstruction_ID)
REFERENCES test_OtherConstruction (ID);

ALTER TABLE test_Other_to_thema_surfa
  ADD CONSTRAINT test_Othe_to_them_sur_FK2 FOREIGN KEY (thematic_surface_ID)
REFERENCES thematic_surface (ID);

-----
-- test_building
-----

```

```

ALTER TABLE test_building
  ADD CONSTRAINT test_building_FK FOREIGN KEY (ID) REFERENCES building (ID);

--
+++++
+++++
-- ***** Create Indexes
*****
--
+++++
+++++

SET SERVEROUTPUT ON
SET FEEDBACK ON
SET VER OFF
VARIABLE SRID NUMBER;
BEGIN
  SELECT SRID INTO :SRID FROM DATABASE_SRS;
END;
/

column mc new_value SRSNO print
select :SRID mc from dual;

prompt Used SRID for spatial indexes: &SRSNO

-----
-- test_BuildingUnit
-----

CREATE INDEX test_Building_Objectc_FKX ON test_BuildingUnit (OBJECTCLASS_ID);

CREATE INDEX test_BuildingU_Parent_FKX ON test_BuildingUnit (BuildingUnit_Parent_ID);

CREATE INDEX test_BuildingUni_Root_FKX ON test_BuildingUnit (BuildingUnit_Root_ID);

CREATE INDEX test_Build_build_buil_FKX ON test_BuildingUnit (building_buildingUnit_ID);

DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='TEST_BUILDINGUNIT' AND
COLUMN_NAME='LOD2MULTICURVE';
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES ('test_BuildingUnit', 'lod2MultiCurve',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0.000, 1000000.000, 0.0005),
MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 1000000.000, 0.0005),MDSYS.SDO_DIM_ELEMENT('Z',
-1000, 10000, 0.0005)), &SRSNO);
CREATE INDEX test_Building_lod2Mul_SPX ON test_BuildingUnit (lod2MultiCurve)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='TEST_BUILDINGUNIT' AND
COLUMN_NAME='LOD3MULTICURVE';
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES ('test_BuildingUnit', 'lod3MultiCurve',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0.000, 1000000.000, 0.0005),
MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 1000000.000, 0.0005),MDSYS.SDO_DIM_ELEMENT('Z',
-1000, 10000, 0.0005)), &SRSNO);
CREATE INDEX test_Building_lod3Mul_SPX ON test_BuildingUnit (lod3MultiCurve)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='TEST_BUILDINGUNIT' AND
COLUMN_NAME='LOD4MULTICURVE';
INSERT INTO USER_SDO_GEOM_METADATA (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES ('test_BuildingUnit', 'lod4MultiCurve',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0.000, 1000000.000, 0.0005),
MDSYS.SDO_DIM_ELEMENT('Y', 0.000, 1000000.000, 0.0005),MDSYS.SDO_DIM_ELEMENT('Z',
-1000, 10000, 0.0005)), &SRSNO);
CREATE INDEX test_Building_lod4Mul_SPX ON test_BuildingUnit (lod4MultiCurve)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

CREATE INDEX test_Building_lod1Mul_FKX ON test_BuildingUnit (lod1MultiSurface_ID);

```

```

CREATE INDEX test_Building_lod2Mul_FKX ON test_BuildingUnit (lod2MultiSurface_ID);
CREATE INDEX test_Building_lod3Mul_FKX ON test_BuildingUnit (lod3MultiSurface_ID);
CREATE INDEX test_Building_lod4Mul_FKX ON test_BuildingUnit (lod4MultiSurface_ID);
CREATE INDEX test_Building_lod1Sol_FKX ON test_BuildingUnit (lod1Solid_ID);
CREATE INDEX test_Building_lod2Sol_FKX ON test_BuildingUnit (lod2Solid_ID);
CREATE INDEX test_Building_lod3Sol_FKX ON test_BuildingUnit (lod3Solid_ID);
CREATE INDEX test_Building_lod4Sol_FKX ON test_BuildingUnit (lod4Solid_ID);

-----
-- test_EnergyPerformanceCer
-----
CREATE INDEX test_Energ_Build_ener_FKX ON test_EnergyPerformanceCer (BuildingU-
ni_energyPerf_ID);

-----
-- test_Facilities
-----
CREATE INDEX test_Faciliti_Objectc_FKX ON test_Facilities (OBJECTCLASS_ID);
CREATE INDEX test_Facil_Build_equi_FKX ON test_Facilities (BuildingU-
ni_equippedWi_ID);

--
+++++
+++++
-- ***** Create Sequences
*****
--
+++++
+++++

CREATE SEQUENCE test_EnergyPerformanc_SEQ INCREMENT BY 1 START WITH 1 MINVALUE 1
CACHE 10000;

```

## Appendix 4: SQL Definition of the TestADE PostGIS DB-Schema

The following PostGIS-compliant SQL definition of the TestADE (cf. chapter 4.3.3) has been automatically generated from its XML schema definition file using the developed graph-based model transformation tool (cf. chapter 4.3.1).

```
-- This document was automatically created by the ADE-Manager tool of 3DCityDB
-- (https://www.3dcitydb.org) on 2017-09-20 11:50:06
--
+++++
+++++
-- ***** Create tables
-- *****
--
+++++
+++++
-----
-- test_BuildingU_to_address
-----
CREATE TABLE test_BuildingU_to_address
(
    BuildingUnit_ID INTEGER NOT NULL,
    address_ID INTEGER NOT NULL,
    PRIMARY KEY (BuildingUnit_ID, address_ID)
);
-----
-- test_BuildingUnit
-----
CREATE TABLE test_BuildingUnit
(
    ID INTEGER NOT NULL,
    OBJECTCLASS_ID INTEGER,
    BuildingUnit_Parent_ID INTEGER,
    BuildingUnit_Root_ID INTEGER,
    building_buildingUnit_ID INTEGER,
    class_codespace VARCHAR(254),
    class VARCHAR(254),
    usage_codespace VARCHAR(254),
    usage VARCHAR(254),
    function_codespace VARCHAR(254),
    function VARCHAR(254),
    lod2MultiCurve geometry(GEOMETRYZ),
    lod3MultiCurve geometry(GEOMETRYZ),
    lod4MultiCurve geometry(GEOMETRYZ),
    lod1MultiSurface_ID INTEGER,
    lod2MultiSurface_ID INTEGER,
    lod3MultiSurface_ID INTEGER,
    lod4MultiSurface_ID INTEGER,
    lod1Solid_ID INTEGER,
    lod2Solid_ID INTEGER,
    lod3Solid_ID INTEGER,
    lod4Solid_ID INTEGER,
    PRIMARY KEY (ID)
);
-----
-- test_EnergyPerformanceCer
-----
CREATE TABLE test_EnergyPerformanceCer
(
    ID INTEGER NOT NULL,
    BuildingUni_energyPerf_ID INTEGER,
    certificationName VARCHAR(254),
    certificationid VARCHAR(254),
    PRIMARY KEY (ID)
);
```



```
-----  
-- test_Facilities  
-----  
CREATE TABLE test_Facilities  
(  
  ID INTEGER NOT NULL,  
  OBJECTCLASS_ID INTEGER,  
  BuildingUni_equippedWi_ID INTEGER,  
  totalValue_uom VARCHAR(254),  
  totalValue NUMERIC,  
  PRIMARY KEY (ID)  
);  
-----  
-- test_IndustrialBuilding  
-----  
CREATE TABLE test_IndustrialBuilding  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR(254),  
  PRIMARY KEY (ID)  
);  
-----  
-- test_IndustrialBuildingPa  
-----  
CREATE TABLE test_IndustrialBuildingPa  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR(254),  
  PRIMARY KEY (ID)  
);  
-----  
-- test_IndustrialBuildingRo  
-----  
CREATE TABLE test_IndustrialBuildingRo  
(  
  ID INTEGER NOT NULL,  
  remark VARCHAR(254),  
  PRIMARY KEY (ID)  
);  
-----  
-- test_OtherConstruction  
-----  
CREATE TABLE test_OtherConstruction  
(  
  ID INTEGER NOT NULL,  
  PRIMARY KEY (ID)  
);  
-----  
-- test_Other_to_thema_surfa  
-----  
CREATE TABLE test_Other_to_thema_surfa  
(  
  OtherConstruction_ID INTEGER NOT NULL,  
  thematic_surface_ID INTEGER NOT NULL,  
  PRIMARY KEY (OtherConstruction_ID, thematic_surface_ID)  
);  
-----  
-- test_building  
-----  
CREATE TABLE test_building  
(  
  ID INTEGER NOT NULL,  
  ownerName VARCHAR(254),  
  floorArea_uom VARCHAR(254),  
  floorArea NUMERIC,
```

```

EnergyPerfor_certificatio VARCHAR (254),
EnergyPerfo_certificati_1 VARCHAR (254),
PRIMARY KEY (ID)
);

--
+++++
+++++
-- ***** Create foreign keys
*****
--
+++++
+++++
-----
-- test_BuildingU_to_address
-----
ALTER TABLE test_BuildingU_to_address
ADD CONSTRAINT test_Buildi_to_adres_FK1 FOREIGN KEY (BuildingUnit_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingU_to_address
ADD CONSTRAINT test_Buildi_to_adres_FK2 FOREIGN KEY (address_ID) REFERENCES
address (ID);

-----
-- test_BuildingUnit
-----
ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_Objectcl_FK FOREIGN KEY (OBJECTCLASS_ID)
REFERENCES objectclass (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUnit_FK FOREIGN KEY (ID) REFERENCES cityobject
(ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUn_Parent_FK FOREIGN KEY (BuildingUnit_Parent_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_BuildingUnit_Root_FK FOREIGN KEY (BuildingUnit_Root_ID)
REFERENCES test_BuildingUnit (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Build_build_build_FK FOREIGN KEY (building_buildingUnit_ID)
REFERENCES test_building (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod1Mult_FK FOREIGN KEY (lod1MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod2Mult_FK FOREIGN KEY (lod2MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod3Mult_FK FOREIGN KEY (lod3MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod4Mult_FK FOREIGN KEY (lod4MultiSurface_ID)
REFERENCES SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod1Soli_FK FOREIGN KEY (lod1Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
ADD CONSTRAINT test_Building_lod2Soli_FK FOREIGN KEY (lod2Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

```

```

ALTER TABLE test_BuildingUnit
  ADD CONSTRAINT test_Building_lod3Soli_FK FOREIGN KEY (lod3Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

ALTER TABLE test_BuildingUnit
  ADD CONSTRAINT test_Building_lod4Soli_FK FOREIGN KEY (lod4Solid_ID) REFERENCES
SURFACE_GEOMETRY (ID);

-----
-- test_EnergyPerformanceCer
-----
ALTER TABLE test_EnergyPerformanceCer
  ADD CONSTRAINT test_Energ_Build_energ_FK FOREIGN KEY (BuildingU-
ni_energPerf_ID) REFERENCES test_BuildingUnit (ID);

-----
-- test_Facilities
-----
ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Faciliti_Objectcl_FK FOREIGN KEY (OBJECTCLASS_ID)
REFERENCES objectclass (ID);

ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Facilities_FK FOREIGN KEY (ID) REFERENCES cityobject (ID);

ALTER TABLE test_Facilities
  ADD CONSTRAINT test_Facil_Build_equip_FK FOREIGN KEY (BuildingU-
ni_equippedWi_ID) REFERENCES test_BuildingUnit (ID);

-----
-- test_IndustrialBuilding
-----
ALTER TABLE test_IndustrialBuilding
  ADD CONSTRAINT test_IndustrialBuildin_FK FOREIGN KEY (ID) REFERENCES building
(ID);

-----
-- test_IndustrialBuildingPa
-----
ALTER TABLE test_IndustrialBuildingPa
  ADD CONSTRAINT test_IndustrialBuild_FK_1 FOREIGN KEY (ID) REFERENCES building
(ID);

-----
-- test_IndustrialBuildingRo
-----
ALTER TABLE test_IndustrialBuildingRo
  ADD CONSTRAINT test_IndustrialBuild_FK_2 FOREIGN KEY (ID) REFERENCES themat-
ic_surface (ID);

-----
-- test_OtherConstruction
-----
ALTER TABLE test_OtherConstruction
  ADD CONSTRAINT test_OtherConstruction_FK FOREIGN KEY (ID) REFERENCES cityobject
(ID);

-----
-- test_Other_to_thema_surfa
-----
ALTER TABLE test_Other_to_thema_surfa
  ADD CONSTRAINT test_Othe_to_them_sur_FK1 FOREIGN KEY (OtherConstruction_ID)
REFERENCES test_OtherConstruction (ID);

ALTER TABLE test_Other_to_thema_surfa
  ADD CONSTRAINT test_Othe_to_them_sur_FK2 FOREIGN KEY (thematic_surface_ID)
REFERENCES thematic_surface (ID);

-----
-- test_building
-----

```

```

ALTER TABLE test_building
  ADD CONSTRAINT test_building_FK FOREIGN KEY (ID) REFERENCES building (ID);

--
+++++
-- ***** Create Indexes
-- *****
--
+++++
-----
-- test_BuildingUnit
-----

CREATE INDEX test_Building_Objectc_FKX ON test_BuildingUnit
  USING btree
  (
    OBJECTCLASS_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_BuildingU_Parent_FKX ON test_BuildingUnit
  USING btree
  (
    BuildingUnit_Parent_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_BuildingUni_Root_FKX ON test_BuildingUnit
  USING btree
  (
    BuildingUnit_Root_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Build_build_buil_FKX ON test_BuildingUnit
  USING btree
  (
    building_buildingUnit_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod2Mul_SPX ON test_BuildingUnit
  USING gist
  (
    lod2MultiCurve
  );

CREATE INDEX test_Building_lod3Mul_SPX ON test_BuildingUnit
  USING gist
  (
    lod3MultiCurve
  );

CREATE INDEX test_Building_lod4Mul_SPX ON test_BuildingUnit
  USING gist
  (
    lod4MultiCurve
  );

CREATE INDEX test_Building_lod1Mul_FKX ON test_BuildingUnit
  USING btree
  (
    lod1MultiSurface_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod2Mul_FKX ON test_BuildingUnit
  USING btree
  (
    lod2MultiSurface_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod3Mul_FKX ON test_BuildingUnit
  USING btree
  (

```

```

    lod3MultiSurface_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod4Mul_FKX ON test_BuildingUnit
  USING btree
  (
    lod4MultiSurface_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod1Sol_FKX ON test_BuildingUnit
  USING btree
  (
    lod1Solid_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod2Sol_FKX ON test_BuildingUnit
  USING btree
  (
    lod2Solid_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod3Sol_FKX ON test_BuildingUnit
  USING btree
  (
    lod3Solid_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Building_lod4Sol_FKX ON test_BuildingUnit
  USING btree
  (
    lod4Solid_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

-----
-- test_EnergyPerformanceCer
-----
CREATE INDEX test_Energ_Build_ener_FKX ON test_EnergyPerformanceCer
  USING btree
  (
    BuildingUni_energyPerf_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

-----
-- test_Facilities
-----
CREATE INDEX test_Faciliti_Objectc_FKX ON test_Facilities
  USING btree
  (
    OBJECTCLASS_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

CREATE INDEX test_Facil_Build_equi_FKX ON test_Facilities
  USING btree
  (
    BuildingUni_equippedWi_ID ASC NULLS LAST
  ) WITH (FILLFACTOR = 90);

--
+++++
+++++
-- ***** Create Sequences *****
--
+++++
+++++

CREATE SEQUENCE test_EnergyPerformanc_SEQ
  INCREMENT BY 1
  MINVALUE 0
  MAXVALUE 2147483647
  START WITH 1

```

```
CACHE 1  
NO CYCLE  
OWNED BY NONE;
```





## Appendix 5: XML Definition of the TestADE Schema Mapping

The following schema mapping file of the TestADE (cf. chapter 4.3.3) has been automatically generated from its XML schema definition file using the developed graph-based model transformation tool (cf. chapter 4.3.1).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<schemaMapping xmlns="http://www.3dcitydb.org/database/schema/3.0">
  <metadata>
    <name>TestADE</name>
    <version>1.0</version>
    <description>TestADE</description>
    <dbPrefix>test</dbPrefix>
  </metadata>
  <applicationSchemas>
    <schema id="test" xmlPrefix="ade5" isADERoot="true">
      <namespace context="citygml-2.0">http://www.citygml.org/ade/TestADE/1.0</namespace>
    </schema>
  </applicationSchemas>
  <complexType>
    <complexType id="EnergyPerformanceCertificationType"
      table="test_EnergyPerformanceCer" path="EnergyPerformanceCertification"
      schema="test">
      <attribute column="certificationName" type="string"
        path="certificationName" schema="test" />
      <attribute column="certificationid" type="string"
        path="certificationid" schema="test" />
    </complexType>
  </complexType>
  <featureTypes>
    <featureType abstract="true" id="_AbstractBuildingUnitType"
      table="test_BuildingUnit" objectClassId="20000" path="_AbstractBuildingUnit"
      schema="test">
      <extension base="AbstractCityObjectType">
        <join table="cityobject" fromColumn="ID" toColumn="ID"
          toRole="parent" />
      </extension>
      <complexType path="class" schema="test">
        <attribute column="class_codespace" type="string"
          path="@codespace" schema="test" />
        <attribute column="class" type="string" path="."
          schema="test" />
      </complexType>
      <complexType path="usage" schema="test">
        <attribute column="usage_codespace" type="string"
          path="@codespace" schema="test" />
        <attribute column="usage" type="string" path="."
          schema="test" />
      </complexType>
      <complexType path="function" schema="test">
        <attribute column="function_codespace" type="string"
          path="@codespace" schema="test" />
        <attribute column="function" type="string" path="."
          schema="test" />
      </complexType>
      <complexProperty refType="EnergyPerformanceCertificationType"
        path="energyPerformanceCertification" schema="test">
        <join table="test_EnergyPerformanceCer" fromColumn="ID"
          toColumn="BuildingUni_energyPerf_ID" toRole="child" />
      </complexProperty>
      <geometryProperty inlineColumn="lod2MultiCurve"
        type="MultiCurve" path="lod2MultiCurve" schema="test" />
      <geometryProperty inlineColumn="lod3MultiCurve"
        type="MultiCurve" path="lod3MultiCurve" schema="test" />
      <geometryProperty inlineColumn="lod4MultiCurve"
        type="MultiCurve" path="lod4MultiCurve" schema="test" />
      <geometryProperty refColumn="lod1MultiSurface_ID">

```

```

    type="MultiSurface" path="lod1MultiSurface" schema="test" />
<geometryProperty refColumn="lod2MultiSurface_ID"
  type="MultiSurface" path="lod2MultiSurface" schema="test" />
<geometryProperty refColumn="lod3MultiSurface_ID"
  type="MultiSurface" path="lod3MultiSurface" schema="test" />
<geometryProperty refColumn="lod4MultiSurface_ID"
  type="MultiSurface" path="lod4MultiSurface" schema="test" />
<geometryProperty refColumn="lod1Solid_ID" type="AbstractSolid"
  path="lod1Solid" schema="test" />
<geometryProperty refColumn="lod2Solid_ID" type="AbstractSolid"
  path="lod2Solid" schema="test" />
<geometryProperty refColumn="lod3Solid_ID" type="AbstractSolid"
  path="lod3Solid" schema="test" />
<geometryProperty refColumn="lod4Solid_ID" type="AbstractSolid"
  path="lod4Solid" schema="test" />
<featureProperty target="AddressType" path="address"
  schema="test">
  <joinTable table="test_BuildingU_to_address">
    <join table="test_BuildingUnit" fromColumn="BuildingUnit_ID"
      toColumn="ID" toRole="parent" />
    <inverseJoin table="address" fromColumn="address_ID"
      toColumn="ID" toRole="parent" />
  </joinTable>
</featureProperty>
<featureProperty target="FacilitiesType" path="equippedWith"
  schema="test">
  <join table="test_Facilities" fromColumn="ID"
    toColumn="BuildingUni_equippedWi_ID" toRole="child">
    <condition column="objectclass_id" value="{target.objectclass_id}"
      type="integer" />
  </join>
</featureProperty>
<featureProperty target="BuildingUnitPartType"
  path="consistsOf" schema="test">
  <join table="test_BuildingUnit" fromColumn="ID"
    toColumn="BuildingUnit_Parent_ID" toRole="child">
    <condition column="objectclass_id" value="{target.objectclass_id}"
      type="integer" />
    <treeHierarchy rootColumn="BuildingUnit_Root_ID" />
  </join>
</featureProperty>
</featureType>
<featureType abstract="true" id="FacilitiesType" table="test_Facilities"
  objectClassId="20001" path="Facilities" schema="test">
  <extension base="AbstractFeatureType">
    <join table="cityobject" fromColumn="ID" toColumn="ID"
      toRole="parent" />
  </extension>
  <complexContent path="totalValue" schema="test">
    <attribute column="totalValue_uom" type="string" path="@uom"
      schema="test" />
    <attribute column="totalValue" type="double" path="."
      schema="test" />
  </complexContent>
</featureType>
<featureType id="BuildingUnitPartType" table="test_BuildingUnit"
  objectClassId="20002" path="BuildingUnitPart" schema="test">
  <extension base="_AbstractBuildingUnitType" />
</featureType>
<featureType id="BuildingUnitType" table="test_BuildingUnit"
  objectClassId="20003" path="BuildingUnit" schema="test">
  <extension base="_AbstractBuildingUnitType" />
</featureType>
<featureType id="DHWFacilitiesType" table="test_Facilities"
  objectClassId="20004" path="DHWFacilities" schema="test">
  <extension base="FacilitiesType" />
</featureType>
<featureType id="IndustrialBuildingType" table="test_IndustrialBuilding"
  objectClassId="20005" topLevel="true" path="IndustrialBuilding"
  schema="test">
  <extension base="AbstractBuildingType">

```

```

    <join table="building" fromColumn="ID" toColumn="ID"
        toRole="parent" />
</extension>
<attribute column="remark" type="string" path="remark"
    schema="test" />
</featureType>
<featureType id="IndustrialBuildingPartType" table="test_IndustrialBuildingPa"
    objectClassId="20006" path="IndustrialBuildingPart" schema="test">
<extension base="BuildingPartType">
    <join table="building" fromColumn="ID" toColumn="ID"
        toRole="parent" />
</extension>
<attribute column="remark" type="string" path="remark"
    schema="test" />
</featureType>
<featureType id="IndustrialBuildingRoofSurfaceType"
    table="test_IndustrialBuildingRo" objectClassId="20007"
    path="IndustrialBuildingRoofSurface" schema="test">
<extension base="RoofSurfaceType">
    <join table="thematic_surface" fromColumn="ID" toColumn="ID"
        toRole="parent" />
</extension>
<attribute column="remark" type="string" path="remark"
    schema="test" />
</featureType>
<featureType id="LightingFacilitiesType" table="test_Facilities"
    objectClassId="20008" path="LightingFacilities" schema="test">
<extension base="FacilitiesType" />
</featureType>
<featureType id="OtherConstructionType" table="test_OtherConstruction"
    objectClassId="20009" topLevel="true" path="OtherConstruction"
    schema="test">
<extension base="AbstractSiteType">
    <join table="cityobject" fromColumn="ID" toColumn="ID"
        toRole="parent" />
</extension>
<featureProperty target="AbstractBoundarySurfaceType"
    path="boundedBy" schema="test">
<joinTable table="test_Other_to_thema_surfa">
    <join table="test_OtherConstruction" fromColumn="OtherConstruction_ID"
        toColumn="ID" toRole="parent" />
    <inverseJoin table="thematic_surface" fromColumn="thematic_surface_ID"
        toColumn="ID" toRole="parent" />
</joinTable>
</featureProperty>
</featureType>
</featureTypes>
<propertyInjections>
<propertyInjection table="test_building"
    defaultBase="AbstractBuildingType">
<join table="test_building" fromColumn="ID" toColumn="ID"
    toRole="child" />
<attribute column="ownerName" type="string" path="ownerName"
    schema="test" />
<complexType path="floorArea" schema="test">
    <attribute column="floorArea_uom" type="string" path="@uom"
        schema="test" />
    <attribute column="floorArea" type="double" path="."
        schema="test" />
</complexType>
<complexType path="energyPerformanceCertification"
    schema="test">
    <type path="EnergyPerformanceCertification" schema="test">
        <attribute column="EnergyPerfor_certificatio" type="string"
            path="certificationName" schema="test" />
        <attribute column="EnergyPerfo_certificati_1" type="string"
            path="certificationid" schema="test" />
    </type>
</complexType>
<featureProperty target="_AbstractBuildingUnitType"
    path="buildingUnit" schema="test">

```

```
<join table="test_BuildingUnit" fromColumn="ID"
      toColumn="building_buildingUnit_ID" toRole="child">
  <condition column="objectclass_id" value="{target.objectclass_id}"
            type="integer" />
</join>
</featureProperty>
</propertyInjection>
</propertyInjections>
</schemaMapping>
```