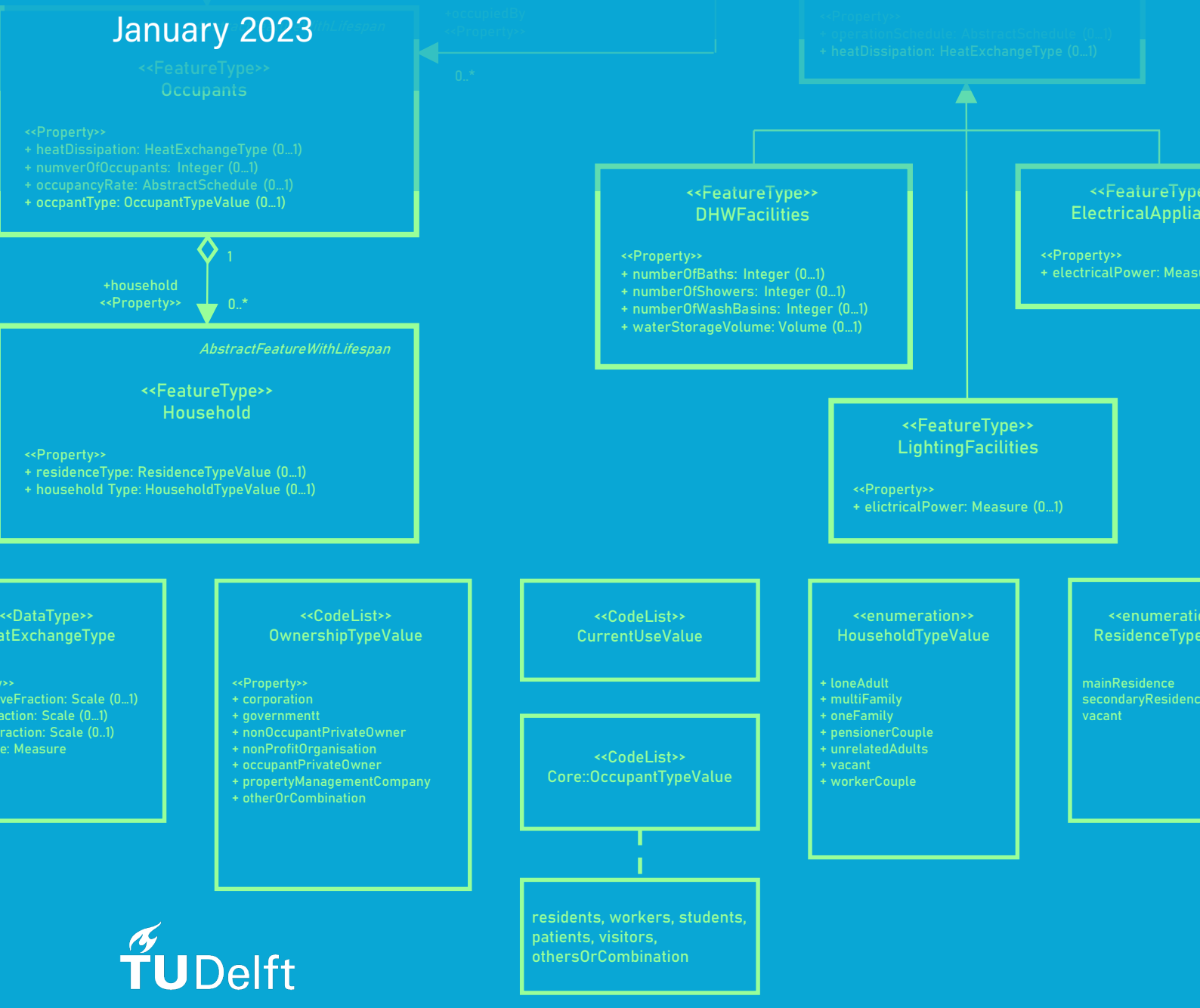
MSc Thesis in Geomatics for the Built Environment

Mapping the Energy ADE to CityGML 3.0

CAROLIN BACHERT

January 2023



TU Delft

MSc thesis in Geomatics

Mapping the Energy ADE to CityGML 3.0

Carolin Bachert

January 2023

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of Master of Science in Geomatics

This work was supervised by:

Delft University of Technology

3D Geoinformation Group

1st supervisor: Dr. Giorgio Agugiaro

2nd supervisor: Camilo León-Sánchez

Technical University of Munich

Chair of Geoinformatics

External supervisor: Dr. Tatjana Kutzner

Stuttgart University of Applied Sciences

Co-reader: Prof. Dr. Volker Coors

Abstract

In order to limit the global warming to well below 2 degrees Celsius, all sectors have to reduce their greenhouse gas emissions and become more sustainable. This also includes the building sector, which is in Europe responsible for 40% of the total energy consumption (European Commission, 2020). A way to work towards this goal is by retrofitting the existing building stock to become more energy efficient. Urban Building Energy Modelling (UBEM) can help in this endeavour by identifying energy-saving potentials and thus to effectively allocate the required resources (Horak et al., 2022). Yet, UBEM involves many stakeholders which is why standards are crucial to facilitate data exchange and interoperability among them. In this context, the Energy ADE v1.0 was developed as an extension for the semantic 3D city model standard CityGML 2.0. It serves two purposes, first by storing energy related information on the individual building level, and second by providing the necessary input data for UBEM simulations (Agugiaro et al., 2018). However, in September 2021 CityGML 3.0 was released. The introduced changes directly affect the structure of the Energy ADE, which is why it cannot fully function on it anymore. This thesis therefore answers the question, how and to what extent the Energy ADE for CityGML 2.0 needs to be adapted to be conformant with the new CityGML 3.0 standard. It is accomplished by following a model-driven approach, where the UML class diagrams for the mapped Energy ADE are created first, before automatically deriving the corresponding XSD schema file. Through the lossless mapping itself, the Energy ADE is integrated as much as possible into CityGML 3.0, while also maintaining a logical symmetry. As such it accounts for the introduced changes of CityGML 3.0, by making use of the space and geometry concept, the versioning possibilities as well as the provided structures to model time-dependent data. The result is eventually tested and verified by converting a sample dataset to the Energy ADE for CityGML 3.0. This work provides an example on how other ADEs can be adapted to fit the new CityGML 3.0 standard and thus hopefully to the further establishment of it.

Acknowledgements

First of all, I would like to thank all my supervisors for their great support, insights and feedback while working on this thesis. During many meetings, Giorgio Agugiaro and Camilo León-Sánchez really helped me to understand the concepts of the Energy ADE and discussed every small detail regarding different mapping possibilities with me. Vital was also Tatjana Kutzner's expert knowledge of CityGML 3.0 and her introductions into Enterprise Architect and ShapeChange.

Furthermore, I want to thank my friends and family who have been there for me in this time, with all their support to keep me motivated but also with sometimes well-deserved distractions. A special thanks goes to my library friends who I have seen day-in and day-out in the last few months, making the long working hours more bearable with all the little coffee breaks in between. Lastly, I am also very thankful for my friends and fellow students from Geomatics who were also in the process of writing their thesis. Because as we say in Germany, "geteiltes Leid ist halbes Leid".

Table of Contents

Figures

Tables

Encodings

Abbreviations

ADE	Application Domain Extension
BIM	Building Information Modeling
CityGML	City Geography Markup Language
HVAC	Heating Ventilation Air Conditioning
IFC	Industry Foundation Classes
LOD	Level Of Detail
OGC	Open Geospatial Consortium
UBEM	Urban Building Energy Modelling
UESM	Urban Energy Systems Modelling
UML	Unified Modeling Language

1. Introduction

Based on the United Nations World Urbanization Prospects, the World Bank estimates that in 2020, 4.36 billion of the world's population lives in urban areas. This makes more than 56% of the people residing in cities, with no prospects of a changing trend (World Bank, 2020a, b). Also the Sustainable Development Goals highlight the multidimensional challenges arising through this development by Goal 11: *Make cities inclusive, safe, resilient and sustainable* (Sustainable Development Goals, 29.11.2022).

Its importance becomes evident through the fact that emissions and resource use are rising with increasing population densities, leading to cities being responsible for over 70% of global carbon emissions (Ribeiro et al., 2019). Within this context, the building sector plays a crucial role. In the European Union, it causes 36% of the greenhouse gas emissions and 40% of the total energy consumption. To strive towards carbon-neutral cities, buildings' energy efficiency consequently needs to be enhanced. A way to achieve this is by renovating and retrofitting the already existing building stock. This measurement could eventually decrease Europe's total energy consumption by up to 6% (European Commission, 2020).

Urban Energy Modelling (UEM) can play a vital role in this endeavour by simulating a city's energy demand and supply. For identifying energy-saving potentials on the building level, bottom-up Urban Building Energy Modelling (UBEM) techniques are particularly suitable (Nageler et al., 2018). Their outcomes can support decision-makers in efficiently allocating the required resources (Horak et al., 2022).

However, various models are in place, differing in their methodology and spatial granularity. Generally, they require a vast amount of input data, ranging from physical building properties, energy consumption, local climate data and occupant behaviour to the buildings' geometries (Corrado & Fabrizio, 2019). This often results in interoperability problems between the different stakeholders (Agugiaro et al., 2018) and thus withholds the potential of UBEM.

To circumvent those issues, domain standards facilitate data exchange and the development of suitable software tools. In regard to model 3D urban environments, only a few open standards exist. At the individual building level, Building Information Modeling (BIM) is frequently implemented through the Industry Foundation Classes (IFC) standard (Agugiaro et al., 2018). The standard has its origin in the field of Architecture and models buildings throughout their lifecycle in high geometric detail (Arroyo Ohori et al., 2022). These characteristics qualify the

standard for precise energy simulations of single buildings. However, at a city-wide level, standards focusing on larger areas and multiple buildings at once, are better suited.

1.1. CityGML and Application Domain Extensions

An open standard designed to operate at this scale is the City Geography Markup Language (CityGML) managed by the Open Geospatial Consortium (OGC) which is currently most widely adopted in version 2.0. It constitutes a storage and exchange format as well as a UML data model for semantically enriched 3D city models, by defining "basic entities, attributes, and relations" (Gröger & Plümer, 2012). The entities refer to in cities occurring dominant features, such as buildings, vegetation or transportation infrastructure. They can geometrically be modelled together with semantic and topographic information in various Levels Of Detail (LOD) (Gröger et al., 2012).

Although CityGML already offers vast possibilities to describe city models, certain applications might require specific classes and attributes that are not foreseen in the standard data model. To account for those cases it is possible to extend the data model in two ways. The first one is through generic attributes and objects, which augment the features' properties and allow to model unrepresented objects respectively. In this way, the added generics are not formally defined by any schema and can be added during runtime. The other possibility is to create specific Application Domain Extensions (ADE) with their own UML class diagram and XSD encoded schema. By this, new feature classes, attributes and relations are formally defined which are embedded in the standard CityGML data model (Gröger et al., 2012). This has the advantage of increased interoperability and the possibility to validate enriched city models (Biljecki et al., 2018).

The Energy ADE v1.0 is exactly such a formal extension of CityGML 2.0. It builds up on the CityGML 2.0 Core and Building modules, by introducing new classes and adding properties to already existing ones. As such, it offers a solution to store UBEM relevant information and serves at the same time as input data for single-building or city-wide energy assessments. Through the standardisation, interoperability between different stakeholders in the domain can be increased (Agugiaro et al., 2018).

Furthermore, the Energy ADE is mentioned as a best practice example in related literature due to the involvement of various international parties in the development, its technical maturity and last but not least its frequent application in research projects (Kolbe et al., 2021) (Biljecki

et al., 2018). For instance Rossknecht & Airaksinen, 2020 developed a concept for heating demand predictions in Helsinki under the different climate change scenarios. Malhotra et al., 2019 scrutinize the approach to enrich the CityGML plus Energy ADE data model through existing simulation platforms depending on different LODs, and Pasquinelli et al., 2019 assess the energy performance of the building stock in Northern Italy. Other examples include (Geiger et al., 2020), (León-Sánchez et al., 2021) and (Coors et al., 2022).

In the meantime, the OGC officially released the new CityGML 3.0 standard in September 2021 (Kolbe et al., 2021). After several years of development in multiple working groups, the final version introduces several changes. Some of the most relevant ones are the revised geometry concept and the newly established space concept in the Core module. Moreover, new modules for man-made constructive elements and time-dependent properties are incorporated (Kutzner et al., 2020). Additionally, the ADE mechanism has been modified for an improved possibility to include several ADEs at the same time (Kolbe et al., 2021).

Those changes directly affect the structure of the Energy ADE. First and foremost, the Energy ADE cannot fully function with CityGML 3.0 due to the new extension mechanism and partially changed class names. Furthermore, some classes and properties in the Energy ADE are now incorporated in the standard CityGML 3.0 data model. Thus, modelling things twice fails to meet the objective of an ADE which is to *extend* where necessary. Beyond that, the OGC standard introduces many additional classes, which potentially offer a better semantic fit to link the ADE classes to.

As CityGML 3.0 is still relatively new, not much research has been published yet in regard to ADEs. Some projects exist which already implement extensions. However, they rather focus on a practical use case with an ADE as by-product, and less on the technical modelling decisions behind their creation. Besides this, the thematically related UtilityNetwork ADE (citygml3-utility-network-ade, 30.11.2022) has been mapped to fit CityGML 3.0. But there is no publication in this regard, describing the process or the modelling decisions behind.

1.2. Research Objective

The objective of this work is thus to map the Energy ADE to CityGML 3.0 ensuring a lossless data conversion and to implement the final result together with a detailed description of the reasonings and processes behind. Within this context, the thesis therefore aims to answer the following research question:

How and to what extent need the Energy ADE for CityGML 2.0 be adapted to be conformant with the newly released CityGML 3.0 standard?

As part of the research, several sub-questions are furthermore specified:

- Which classes of the Energy ADE 1.0 become obsolete, which ones need to be adapted and which ones can mostly be taken over?
- What will the Energy ADE data model for CityGML 3.0 look like, both in terms of UML encoding and XSD file?
- How can Energy ADE for CityGML 2.0 data be converted to Energy ADE for CityGML 3.0 data?

The scope is thus to solely map the existing Energy ADE onto CityGML 3.0, to produce a valid XSD file, and to implement a testing pipeline to convert data losslessly. Its content in terms of semantics and functionalities is not to be altered. Though, some minor changes might inevitably occur due to the new structure of CityGML 3.0.

1.3. Method

The research is accomplished by adapting the model-driven approach as described by van den Brink et al., 2013, to fit CityGML 3.0. First, the UML class diagrams of the Energy ADE 1.0 for CityGML 3.0 are created before automatically deriving the corresponding XSD schema file with the Java tool ShapeChange. Finally, test data is created and subsequently converted to the newly established ADE. This step is implemented in the ETL software FME.

All resources of this thesis are collected in a GitHub repository.

All over, I am aspiring to contribute with this work to the further establishment of CityGML 3.0 as a new standard. This includes to keep a widely used ADE up to date with the most recent developments in the Geoinformation domain and to serve as an information resource for future endeavours in developing the Energy ADE. Moreover, I hope this research provides some useful insights for other practitioners working with ADEs in CityGML 3.0.

1.4. Reading Guide

The remaining thesis is structured as follows.

Chapter 2 gives an extensive insight into relevant concepts, theories and related work. Followingly, Chapter 3 explains the methodology including the applied model-driven approach in more detail, together with the used tools and software. Chapter 4 subsequently describes the implementation of the mapping, the schema file derivation and the data conversion in depth. Chapter 5 presents the obtained results before discussing them in Chapter 6. Finally, Chapter 7 draws the conclusions of this work.

2. Scientific Context

In this chapter some theoretical background is provided. It gives an insight into relevant concepts, data models and methodologies together with related literature. For this, approaches for Urban Building Energy Modelling are briefly explained. Following is an introduction to the basics of Unified Modeling Language as it is inevitable to understand when working with CityGML. Subsequently, relevant concepts of CityGML 2.0 are outlined before describing the Energy ADE 1.0 in detail. Lastly, the main changes of the new CityGML 3.0 standard are presented.

2.1. Urban Building Energy Modelling

Urban Energy Modelling (UEM) frameworks, analysing a city's energy demand and supply (Abbasabadi & Ashayeri, 2019; Horak et al., 2022), and Urban Building Energy Modelling (UBEM), focussing on the building sector (Nageler et al., 2018; Reinhart & Cerezo Davila, 2016), are valuable methods for urban planners and policy makers to work towards more sustainable cities. Generally, UBEM can be distinguished in top-down and bottom-up approaches. Top-down models are based on historic data of the overall energy flow in a certain area. This information can then be further decomposed to match smaller sections. Bottom-up models on the other hand, operate rather on individual or multiple building level and are then scaled up to larger extents. They require detailed information about the objects' energy relevant characteristics, such as energy demand and supply, thermal properties or occupancy behaviour (Abbasabadi & Ashayeri, 2019; Reinhart & Cerezo Davila, 2016).

Bottom-up models can be further subdivided into two main categories, the physical modelling approach, also called simulation-based engineering by Abbasabadi & Ashayeri, 2019, and the data-driven approach (Nageler et al., 2018). Data-driven models establish a mathematical relationship through artificial intelligence or statistical techniques between provided energy datasets and the "energy use of individual end-uses" (Abbasabadi & Ashayeri, 2019). The physical modelling approach relies on a combination of building geometries, their physical characteristics and additional information e.g. regarding their heating, ventilation and air-conditioning (HVAC) systems or internal gains. As these non-geometric information are often difficult to obtain on an individual level, building archetypes are commonly used instead (Abbasabadi & Ashayeri, 2019; Nageler et al., 2018). They describe certain building typologies with shared properties (Reinhart & Cerezo Davila, 2016).

The described bottom-up approaches both require certain energy relevant building information for their implementation. According to Corrado & Fabrizio, 2019, they can be categorised into *Climate* (time-varying information about weather phenomena), *Building* (including information regarding the building envelope with its construction layers and thermal zones), *HVAC* (information about subsystems for energy -emission, -control, -distribution, -storage, -generation) and *Users* (including occupancy schedules and operation schedules for e.g. ventilation, thermostats or shading devices). For each of these aspects exist equivalents in the Energy ADE which are presented in part 2.4.2.

The Energy ADE for CityGML thus closes the gap of an open energy data model at city scale. Open standards are pivotal for a successful establishment of UBEM, as they involve many different actors and stakeholders. Without such standards data interoperability and exchange can be difficult due to non-uniform object definitions and formats. This potentially leads to a loss of information, requires more time and resources to obtain good quality harmonised data and eventually fewer software solutions for UBEM (Agugiaro et al., 2018).

Existing open standards to model the built environment with the possibility to include energy related information are sparse. They moreover vary in their covered geographic scale and level of detail depending on the application domain (Agugiaro et al., 2018). On the individual building level, the IFC data model is commonly used to represent buildings from the architectural and constructional point of view throughout their lifecycle. It provides various entities and attributes to represent the detailed structure of a building (Arroyo Ohori et al., 2022). Also in the world of BIM, the Green Building XML schema (gbXML) "enables interoperability between disparate building design and engineering analysis software tools" (gbXML, 02.01.2023). Based on these characteristics the standard is particularly suited for energy analyses on the building level (Agugiaro et al., 2018). Opposed to this, the INSPIRE Specification on Buildings operates on the cross country and international level within the European Union. It provides attributes to describe, among others, a buildings' heating system, its energy performance or network connections. However, it fails to meet the complexity needed for UBEM (INSPIRE Thematic Working Group Buildings, 2013; Agugiaro et al., 2018).

The Energy ADE for CityGML facilitates UBEM on the level between, on the city-wide scale. As such, is shares some overlap on both ends with BIM and ISPIRE (Figure 1). In order to continue with more detailed explanations of CityGML and the Energy ADE, the basics of UML described in the following.

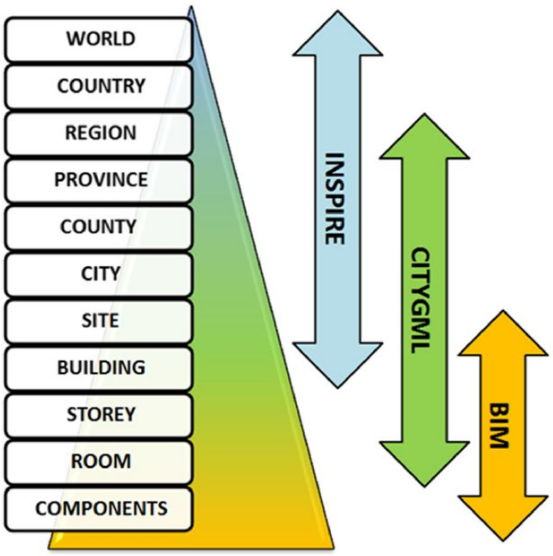


Figure 1: Applicability of open building standards on a geographical scale. Taken from Agugiaro et al., 2018.

2.2. Introduction to UML

As of version 3.0, the CityGML Conceptual Model and its ADEs are formally defined through Unified Modeling Language (UML) object models. Also in the earlier versions, UML plays a crucial role in specifying the standard and exchange format. It thus constitutes an integral part of this thesis and is followingly explained in its fundamental structures.

UML was first introduced in 1995 by G. Booch, J. Raumbaugh and later on I. Jacobson as a tool for "*specifying, visualizing and documenting object-oriented systems*" (Breu et al., 1997). Since then it has been standardised through the Object Management Group (OMG) in 1997 (Rumbaugh et al., 1999; *UML v2.5.1*, 2017) and is available as ISO standard in version UML 2.4.1 (*ISO/IEC 19505-2:2012*, 2012).

The UML subsumes several kinds of diagram types, which are classified according to so-called *views*. In the case of CityGML, *class diagrams* are used. They describe a static system consisting of objects and their (static) relationships to each other (Rumbaugh et al., 1999). The for CityGML relevant concepts of this diagram type are further explained in more detail, whereas the descriptions are mostly based on Pilone & Pitman, 2005 and own experience.

Classes, Objects and Features

At the core of the class diagram are the *classes*. They represent a group of objects with common characteristics and behaviour. An *object* or *feature* on the other hand, is a concrete instance of

this class (Pilone & Pitman, 2005). An exemplary class could be "*Tree*", with instances being a specific oak or a maple tree feature.

Attributes

Classes can have several *attributes* giving additional information about its instances. In the context of CityGML and according to General Feature Model defined in ISO 19109, they are also called *properties* (*ISO 19109:2015*, 2015). Within the tree example, this could be the species of the tree, its height and age. Each attribute is of a given *type*, which can be simple (e.g. CharacterString, Double) or complex (e.g. an individually defined data type).

Multiplicity

Attributes and relations (see next point) have a *multiplicity*. In the context of attributes, it specifies how many instances of this attribute are required or allowed per object. For relationships, it defines the number of instances of another class an object can relate to. Multiplicities are defined as follows:

Integer	[2]	Requires exactly two instances
Range	[0..1]	Requires number of instances within the given range
	[0..*]	The asterisk symbolises an indefinite number
Missing		Requires exactly one instance

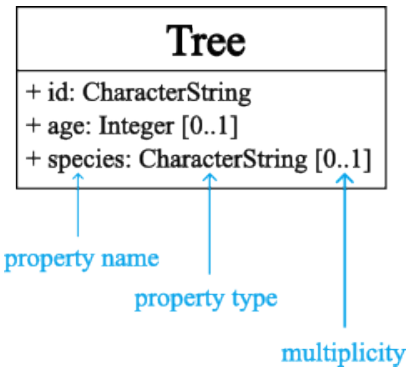


Figure 2: UML class with several properties.

Relations

Classes can be in relation with each other. The way the *relations* are drawn in the UML class diagram, gives information about the kind of relationship. Except for inheritance, relations also have multiplicities and *role names*. The role name gives a semantic meaning to the relation between two classes.

Association

Associations express a general relationship between classes which can be parsed in both directions. Additional context is given through the role name and the multiplicity. In essence, such a relationship is the same as an inline attribute of a class, where the role name corresponds to the attribute name and the other class to the data type. Therefore, these two concepts can be used interchangeably.

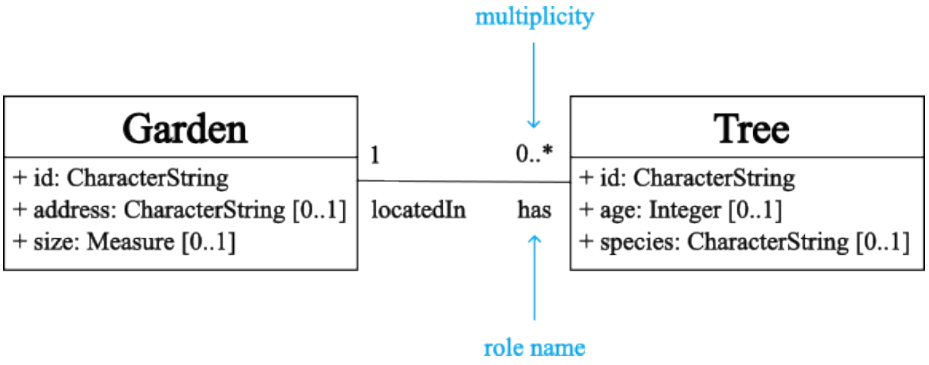


Figure 3: Association between two classes. A Garden has zero to many Trees, whereas a Tree is locatedIn only one Garden.

Associations that are only parsed in one direction, are called *unidirectional associations*. The navigability is indicated with an arrow towards the class to navigate to. If both ends of the relationship have an arrow, it is a *bidirectional association* thus the same as a regular association described before.

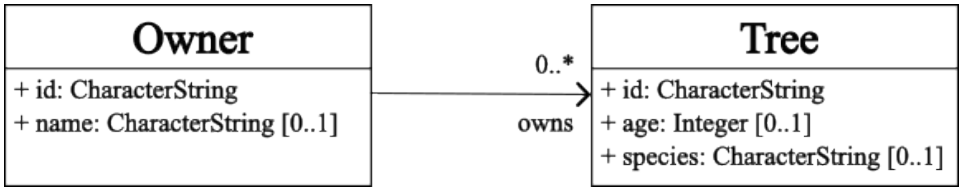


Figure 4: Navigable association between two classes.

Aggregation

An *aggregation relationship* expresses an ownership of one class over the other. However, the classes may still exist independently. It can be read as "*class A owns class B*". In UML diagrams, it is represented through an empty diamond shape at the owning class.

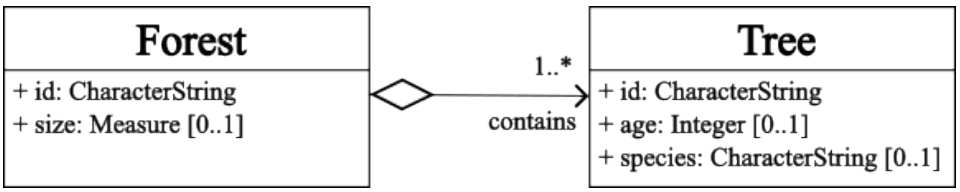


Figure 5: Aggregation relationship. A Forest contains one to many Trees.

Composition

Composition relationships are similar to aggregations but constitute a stronger relationship between the classes. It is read as "*class A contains class B*" or "*class B is a part of class A*". Thus, the contained class cannot exist on its own. It is drawn with a filled diamond shape at the owning class.

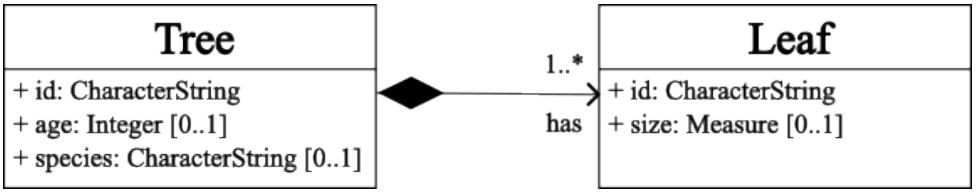


Figure 6: Composition relationship. A Tree contains 1 to many Leaves.

Generalisation or inheritance

As the name already suggests, a *generalisation relationship* is between a general parent class and one or multiple more specific child classes. The parent class subsumes properties common to the child elements. Like this, all these properties, as well as the relations from and to the parent class, are inherited by the child elements. Therefore, this kind of relationship is also called *inheritance*. It is generally used to emphasize the commonality between the child classes. Generalisation relationships are depicted through a closed arrow, pointing towards the parent class.

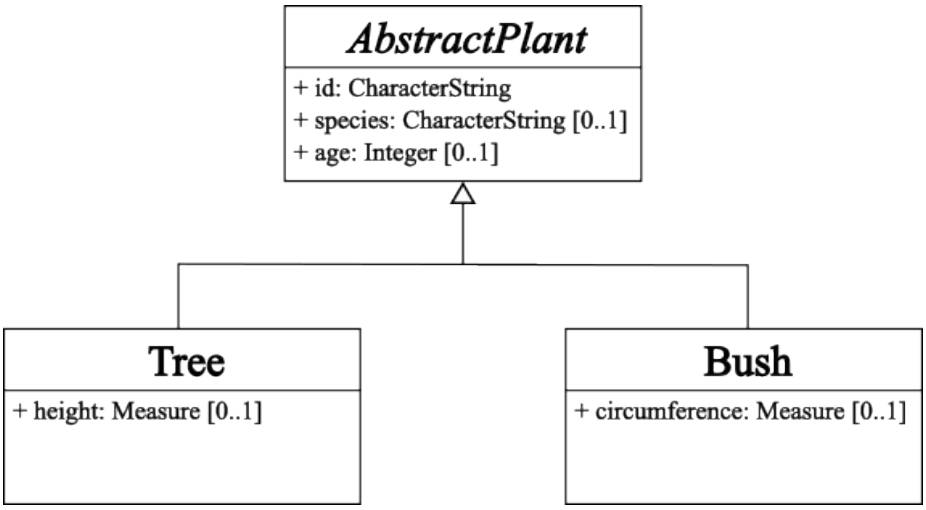


Figure 7: Generalisation relationship. An AbstractPlant can be a Tree or a Bush.

A concept commonly used within CityGML in this context, is the one of *abstract classes*. An abstract class is a parent class which never has an instance of itself. Only the child elements can instantiate objects. A specific example from CityGML 3.0 is the *AbstractConstructionSurface* class with several kinds of construction surfaces, such as *RoofSurface*, *GroundSurface* or *WallSurface*, as child elements.

Stereotypes

Each element in the UML class diagram may be given a *stereotype*. It defines the role of an element, for example a feature type or a data type. Stereotypes are depicted above the class or element name in angle brackets («») (Pilone & Pitman, 2005). The different kinds of stereotypes used in CityGML 3.0 and the Energy ADE are described in detail in chapter 5.2 of the CityGML Conceptual Model Standard (Kolbe et al., 2021).

2.3. CityGML 2.0

The Energy ADE in its current version (v1.0) was developed for the CityGML 2.0 standard published in 2012. However, the first version of the data model for representing and exchanging 3D city models dates back to 2008. Since then, it formed the basis of numerous applications and research projects worldwide (Gröger et al., 2012). This is likely due to its unique approach of defining the "three-dimensional geometry, topology, semantics and appearance of the most relevant topographic objects in urban or regional contexts" all within one open data model (Gröger & Plümer, 2012).

This is implemented by specifying common entities, attributes and relations, organised in 13 thematic modules, connected with each other through the Core module. Here, the concept of city objects plays an important role. They describe in cities occurring dominant features such as buildings, bridges or trees, all deriving from the abstract class *_CityObject* defined in the Core module (Gröger et al., 2012).

Moreover, features in CityGML are geometrically modelled through boundary representation, meaning that solid objects are defined over their bounding surfaces. In the case of *Building* in the Building module, this can be implemented with its corresponding *RoofSurface*, *WallSurface* and *GroundSurface*. Additionally, each feature can be modelled in five different LODs, with increasing semantic and geometric richness. LOD0 refers to the feature's 2.5D footprint. LOD1 is a prismatic block representation, often obtained by extruding LOD0. Next, LOD2 includes in the case of *Building*, a simplified roof shape and the possibility to model it through its thematic surfaces. LOD3 then represents the feature's most detailed shape from the outside. For *Building*, this includes openings such as windows and doors. Lastly, LOD4 adds the interior of the feature to it (Biljecki et al., 2016; Gröger & Plümer, 2012). Figure 8 shows the LODs on the example of a building.

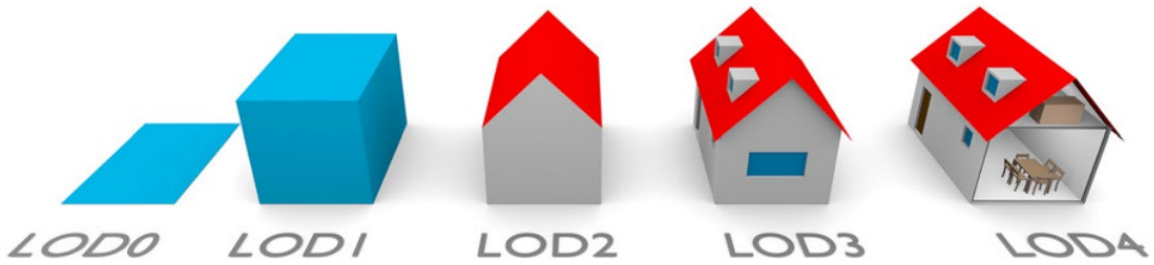


Figure 8: The five LODs of CityGML 2.0 on the example of a Building. Taken from Biljecki et al., 2016.

Although the data model of CityGML is extensive, it is impossible to include all possible features and attributes that might be needed for specific applications. Therefore, two ways of extending CityGML 2.0 are foreseen.

The first one is through adding generic city objects (*GenericCityObject*) and generic attributes (*_genericAttribute*) on demand. They are defined in the Generics module and can be added in runtime without extending the CityGML 2.0 XSD schema. While this is a convenient way of augmenting the data model with simple features, it does not suffice to model more complex systems, nor is it possible to validate the additional information against the given schema (Biljecki et al., 2018; Gröger & Plümer, 2012).

The extension through formally defined ADEs closes this gap. They can specify additional features, relations, properties, data types and geometries in a separate XSD schema with its own namespace. In this way, the information can be standardised and validated, ensuring better data interoperability (Biljecki et al., 2018; Gröger & Plümer, 2012). As such, ADEs make use of two extension mechanisms to achieve this:

1. Augment existing CityGML features with new properties and relations through the ADE hook.

   The CityGML classes to be extended are subclassed by a new element with the stereotype «ADEElement». Within this new element, the additional properties can be defined. The specialisation relation itself is marked with the stereotype «ADE» (Figure 9) (van den Brink et al., 2013).

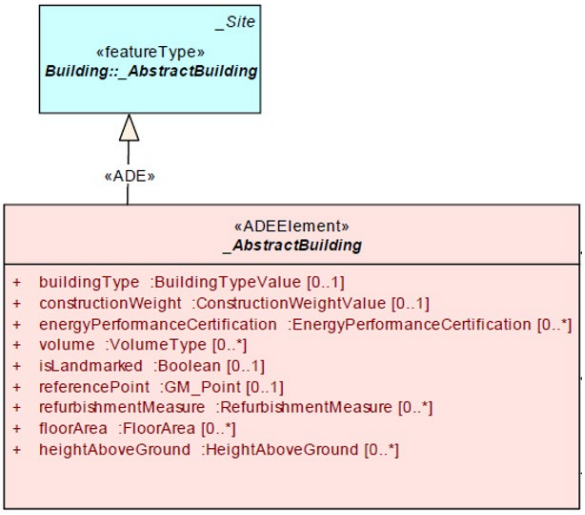


Figure 9: Extending _AbstractBuilding with additional properties via the ADE hook mechanism in the Core module of the Energy ADE 1.0 for CityGML 2.0. Taken from Agugiaro et al., 2018.

2. Define new feature types by subclassing CityGML classes.

   New features can be defined through subclassing *_CityObject* and *_Feature* or a suitable specialisation class thereof. The new feature can then define its own properties and relations (Figure 10) (Biljecki et al., 2018).

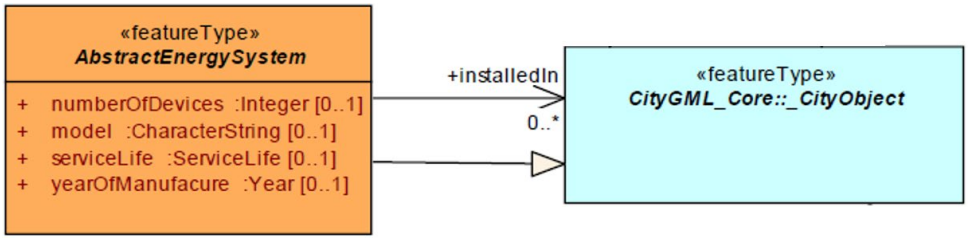


Figure 10: Defining the new feature type AbstractEnergySystem as a subclass of _CityObject in the Energy ADE 1.0 for CityGML 2.0 Core module. Taken from Agugiaro et al., 2018.

2.4. The Energy Application Domain Extension

2.4.1. General

Application Domain Extensions for CityGML are a popular way to include certain features to the data model which are needed for specific applications. This is demonstrated by the amount of different ADEs, which vary greatly in their complexity and maturity. Biljecki et al., 2018 give an extensive overview of existing ADEs and their developments.

The Energy ADE is mentioned in this context as an exemplary development. It started as a joint effort in 2013 of the University of Applied Sciences Stuttgart and the Technical University of Munich to harmonise their data models for urban energy simulations. Followingly, eleven European organisations, among them universities and research institutes, joined the collaboration to further develop the ADE. Additionally, the urban energy simulation platforms SimStadt, CitySIM, Modelica library AixLib, EnergieAltlas, the Curtis platform and SUNSHINE cooperated from the beginning on. This international joint project resulted in the official release of the Energy ADE version 1.0 for CityGML 2.0 in January 2018 (Agugiaro et al., 2018). This current version, including all the recent developments, can be found in a public Git Repository (Energy ADE, 07.11.2022).

The purpose of the Energy ADE is twofold. On the one hand, it serves as an open, standardised data model to “store and manage energy relevant data at urban scale” (Agugiaro et al., 2018). This simultaneously ensures data interoperability between different stakeholders and facilitates the data exchange. On the other hand, it can be used as input for UBEM, both on the single-building as well as the city-wide level. Through its flexibility based on the provided classes and properties, it is applicable for simpler analysis, but also for very detailed complex ones (Agugiaro et al., 2018). An insight in simulation tools supporting the Energy ADE for CityGML 2.0 is given in part 2.4.3.

Since the release of the Energy ADE 1.0 in early 2018, there have been a few suggestions for a further development towards a version 2.0. Some of the proposed changes can be found in the corresponding Git Repository under “Issues” (Energy ADE, 03.06.2022). Furthermore, Dr. Joachim Benner held a presentation about a critical review and proposal of a new Energy ADE model. He mainly points some missing base classes, properties and relations out and subsequently proposes a simpler version of the Energy System module (Energy ADE Review, 03.06.2022). Beyond this, Schildt et al., 2021 suggest to extend the Energy ADE in a new version with additional energy systems and building utilities. However, there has not been any official meeting yet among the working group to discuss the further development of the ADE.

2.4.2. Modules

The Energy ADE data model extends the CityGML Core and Building module by adding new properties to existing classes as well as introducing new feature and data types. The ADE is subdivided into six thematic modules, designed as UML diagrams, with interrelations to one another. In the following, the modules including their most important features are outlined. A more in depth description is given in Agugiaro et al., 2018.

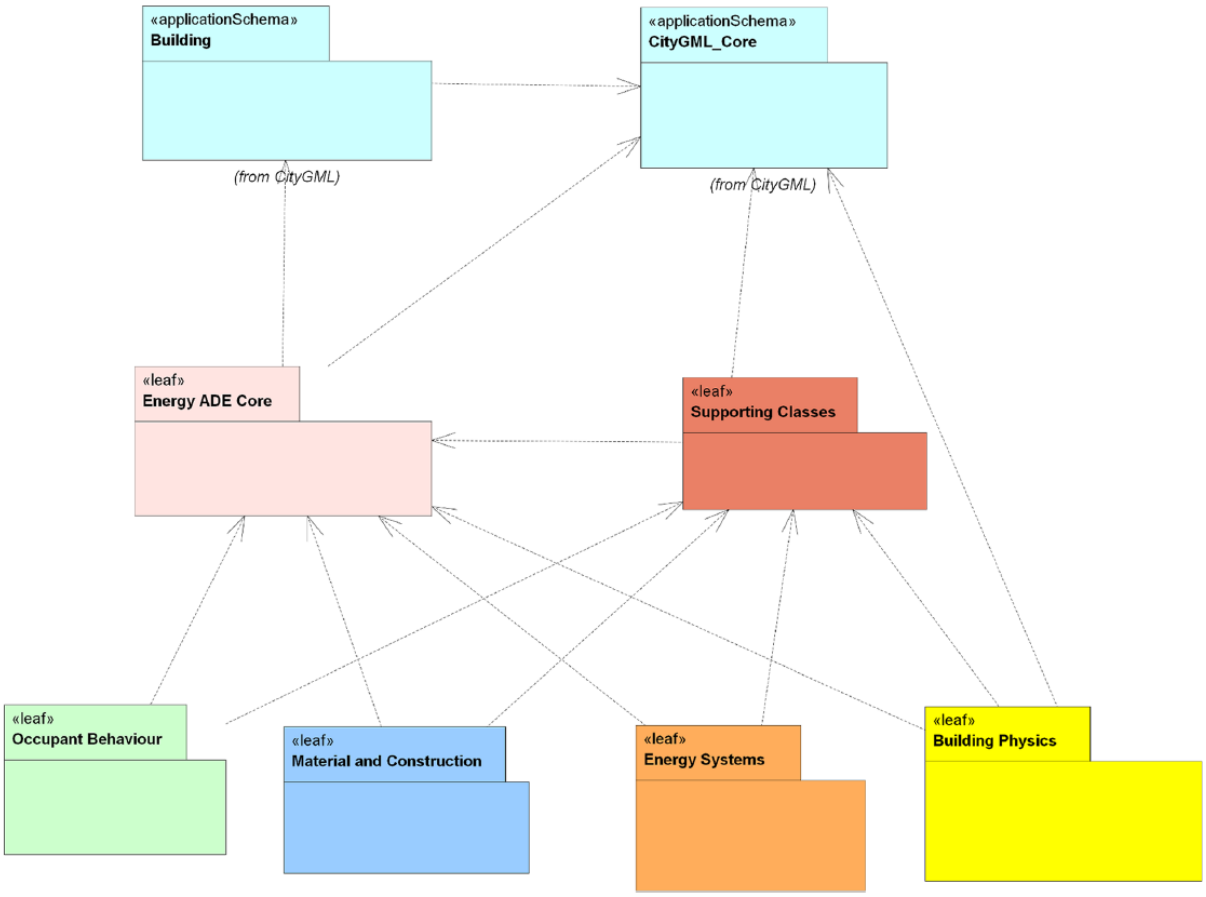


Figure 11: Overview of the modular structure of the Energy ADE for CityGML 2.0 and its interrelations. Taken from Agugiaro et al., 2018.

Core Module

Overall, the Energy ADE Core module defines additional attributes for the *_AbstractBuilding* and *_CityObject* classes, introduces abstract base classes for the other modules and provides additional property types (Figure 12).

The CityGML 2.0 class *_AbstractBuilding* is extended via the ADE hook mechanism to add general parameters for roughly evaluating a building's energy demand. They include additional information regarding the geometry and location (*floorArea*, *volume*, *heightAboveGround*, *referencePoint*) which are not provided by CityGML 2.0, its construction structure (*constructionWeight*) and the type of building (*buildingType*). These added properties can be applied to both, the *Building* and the *BuildingPart* class.

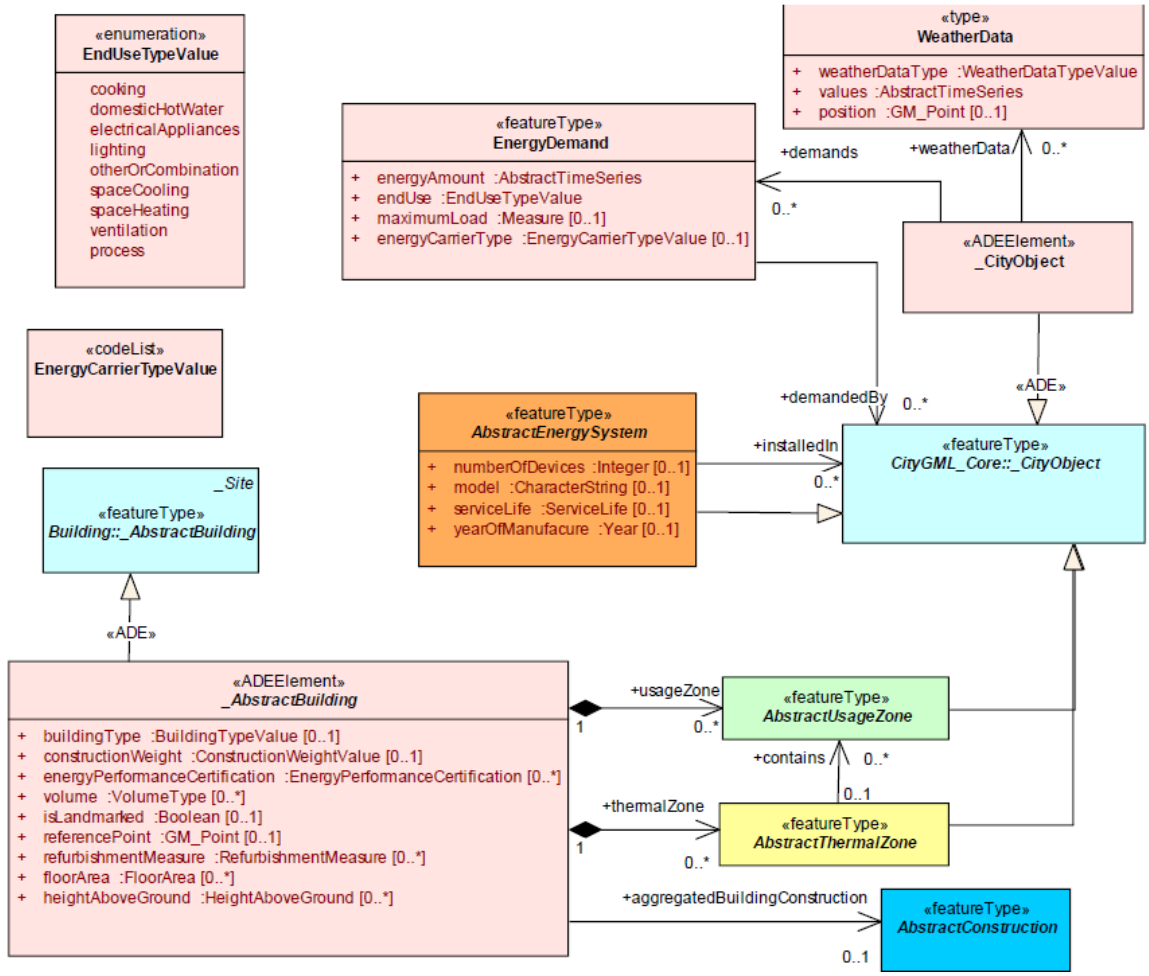


Figure 12: The Energy ADE for CityGML 2.0 Core module.

Furthermore, any *_CityObject* can be extended with the relations to *EnergyDemand* and *WeatherData*. The *EnergyDemand* feature type allows to model the energy demand (*energyAmount*) of various end uses (*endUse*) in a time-depending manner. As such, it serves as a greatly simplified alternative to the more complex *AbstractEnergySystem* classes. For more detailed energy analysis, the *WeatherData* type allows to model time-varying weather phenomena such as temperature and solar irradiance. Because these two classes can be applied to any *_CityObject*, they constitute means to model energy related properties also for features outside of buildings.

Additionally, the abstract classes for the other thematic modules of the Energy ADE (*AbstractThermalZone*, *AbstractUsageZone*, *AbstractConstruction*, *AbstractEnergySystem*) are introduced in the Core. This moreover illustrates their relation to one another. Lastly, additional data types, codelists and enumerations needed for features' properties are established (Agugiaro et al., 2018).

Building Physics Module

For more detailed UBEM, buildings are partitioned in in one or multiple thermal zones to simulate their thermal behaviour. A thermal zone describes an isothermal volume, which can refer to only one room, or for simplified models to an aggregation of rooms up to the whole building. Thermal zones are delimited from each other and the outside of a building via thermal boundaries. They can be defined as surfaces with a uniform thickness and energy transfer rate. Discontinuities within them, typically windows or doors, are called thermal openings.

These concepts are respectively modelled in the Buildings Physics module through the classes *ThermalZone*, *ThermalBoundary* and *ThermalOpening*. It is possible to geometrically represent them, whereby the *ThermalZone* volume needs to be fully enclosed by the *ThermalBoundary* and *ThermalOpening* surfaces.

A *ThermalZone* can contain several *UsageZones*, which hold information about heating-, cooling- and ventilation schedules. *ThermalBoundary* and *ThermalOpening* in turn have a mandatory relation to the *AbstractConstruction* class, specifying optical and thermal properties of the surfaces (Agugiaro et al., 2018). The whole Building Physics module is depicted in Figure 13.

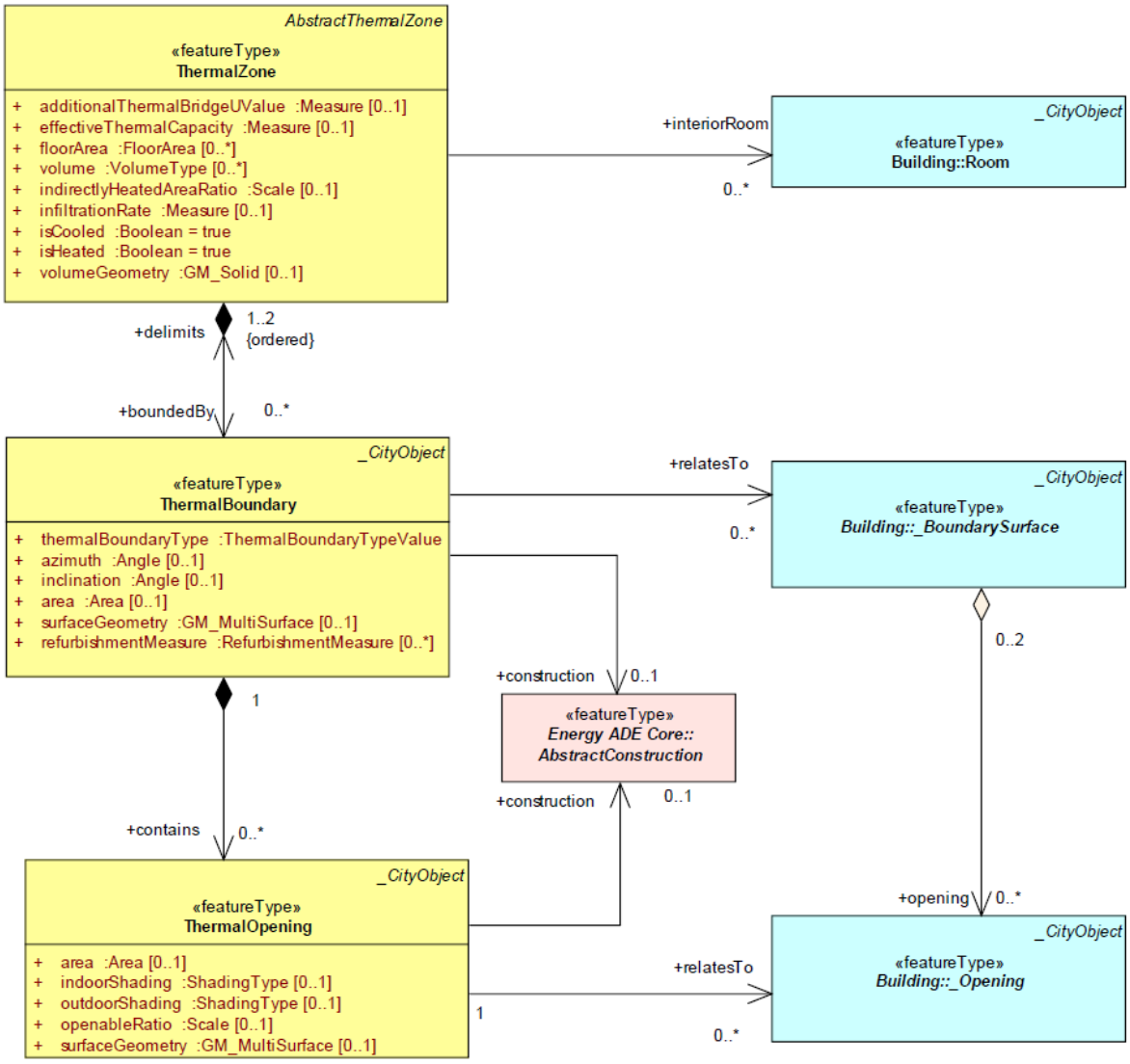


Figure 13: The Energy ADE for CityGML 2.0 Building Physics module.

Material and Construction Module

As mentioned, the Material and Construction module provides classes to remodel construction structures in terms of their optical and thermal properties. This is implemented through a *Construction* class with a property for the given heat transmission coefficient *uValue*. Alternatively, it can also be modelled in more detail by several ordered layers (*Layer*). They are themselves defined through one or multiple *LayerComponent* instances with a certain *thickness*, made of a specified material (*Gas*, *SolidMaterial*). A *Construction* can also be set up as a *ReverseConstruction*, where the order of the layers is inverted (Agugiaro et al., 2018) (Figure 14).

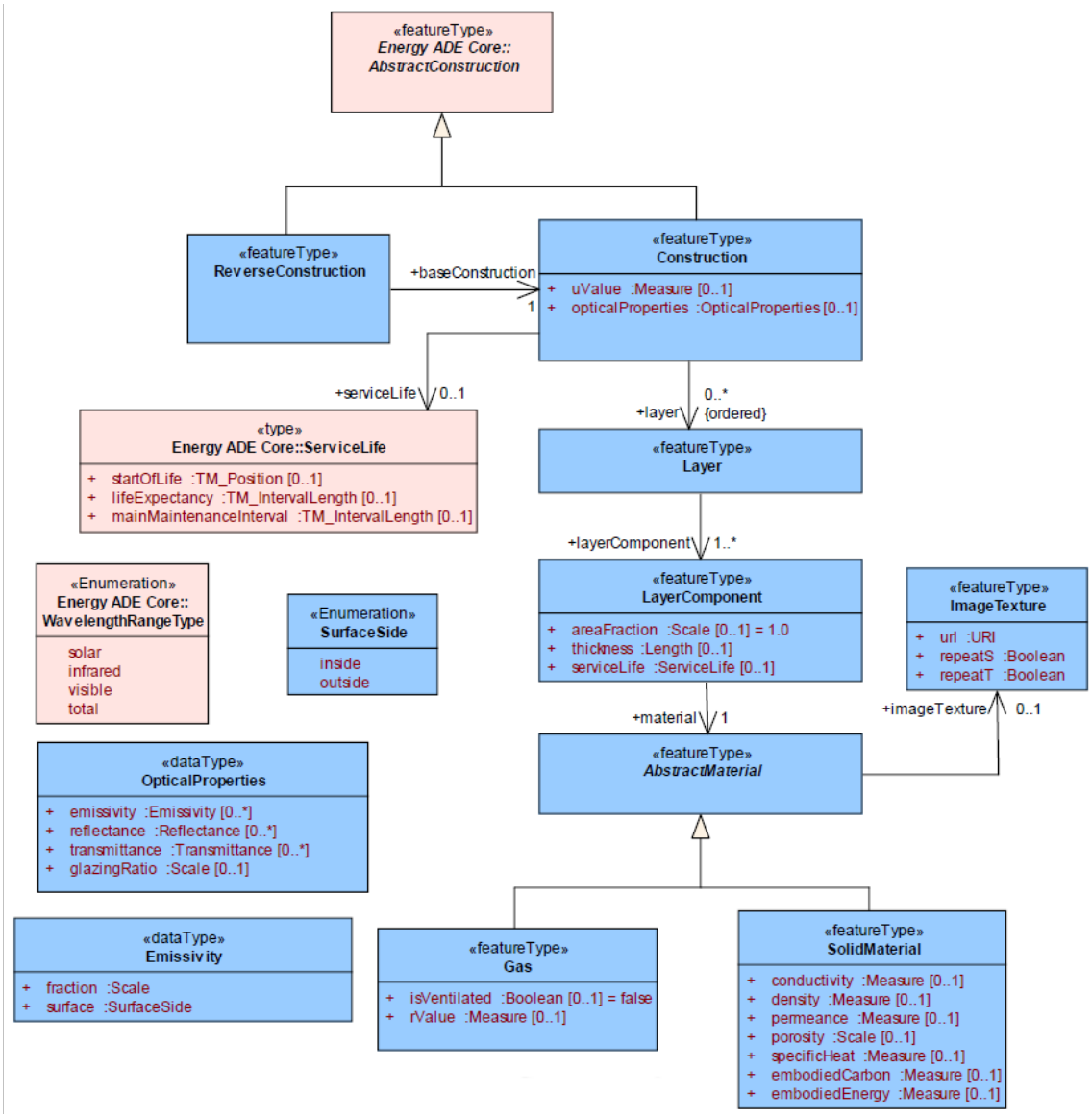


Figure 14: The Energy ADE for CityGML 2.0 Material and Construction module.

Occupant Behaviour Module

The Occupant Behaviour module defines classes to model different usage zones and how they are utilised by occupants and facilities such as electrical appliances. By including schedules it is possible to represent their behaviour over the day/year/etc.

At the core of the module is the class *UsageZone*, which "defines regions of homogenous usage" (Agugiaro et al., 2018). It provides properties to specify the current use (*usageZoneType*) and influences on the indoor temperature (*coolingSchedule*, *heatingSchedule*, *ventilationSchedule*). A *UsageZone* can furthermore contain several building units (*BuildingUnit*), accommodating ownership information. Both the *UsageZone* and the

BuildingUnit may additionally have relations to *Facilities* (including *DHWFacilities*, *LightingFacilities*, *ElectricalAppliances*) and *Occupants* to model internal heat gains produced by them (Agugiaro et al., 2018).

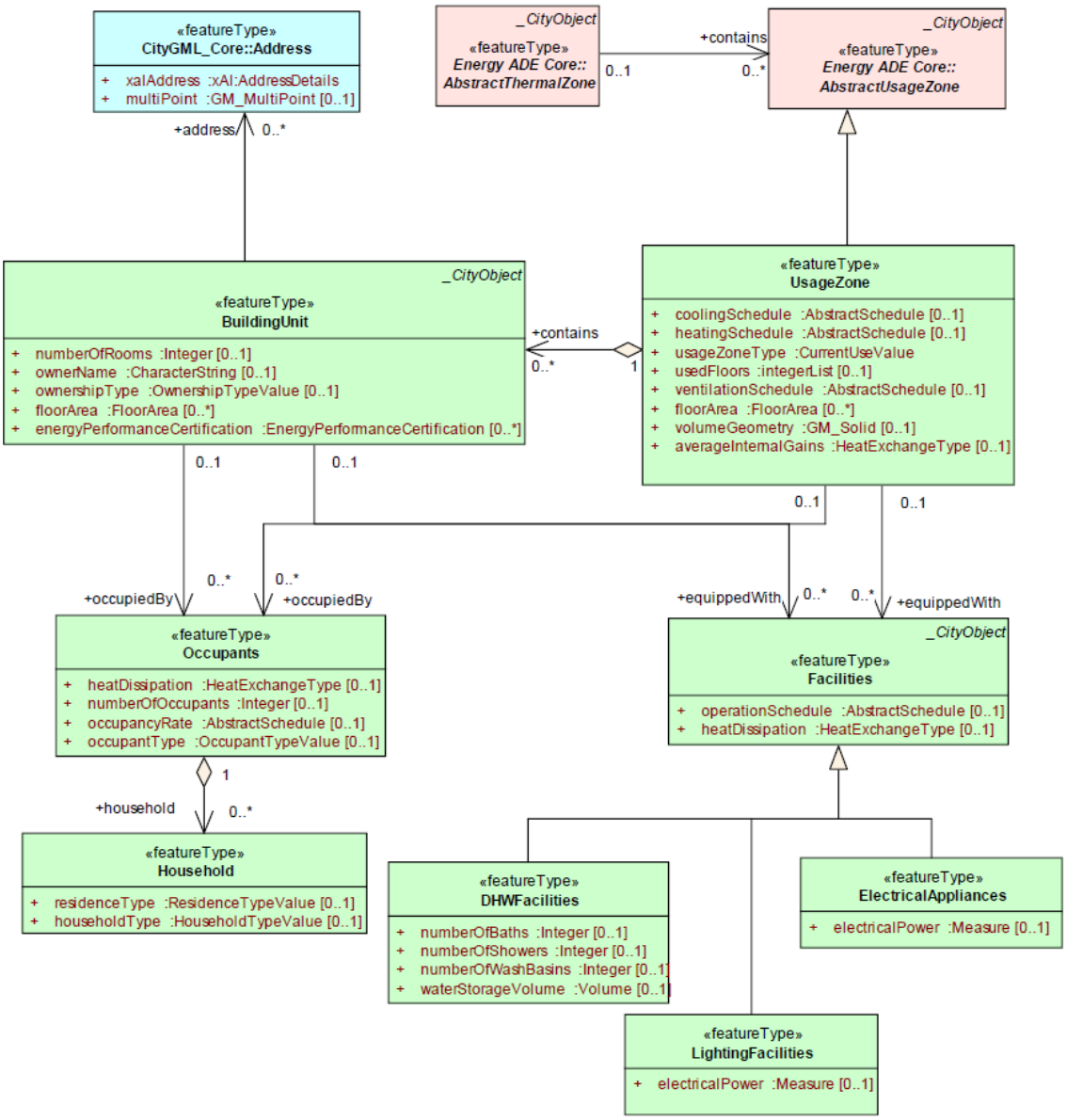


Figure 15: The Energy ADE for CityGML 2.0 Occupant Behaviour module.

Energy Systems Module

For detailed energy supply and demand analyses, additional information regarding a building's energy systems are crucial. These can be provided through *AbstractEnergySystem* and its associated classes in the Energy Systems module. It entails classes to model a building's energy

storage (*AbstractStorageSystem*), distribution (*AbstractEnergyStorageSystem*), emission (*EmitterSystem*) and conversion appliances (*AbstractEnergyDistributionSystem*). The module furthermore showcases how the energy is exchanged and transmitted among them through the *EnergyFlow* with its mandatory property *energyAmount* (Agugiaro et al., 2018) (Figure 16).

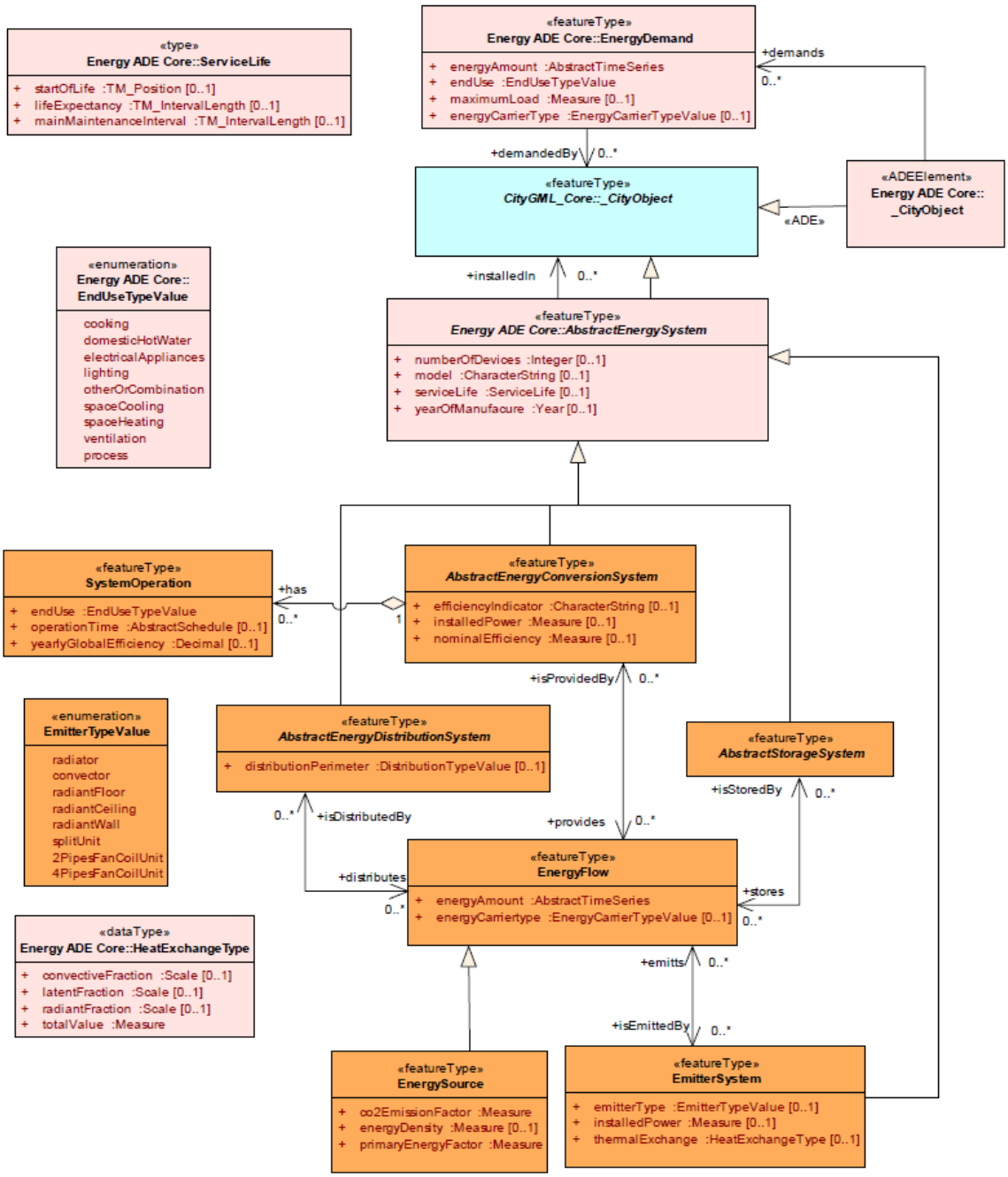


Figure 16: The Energy ADE for CityGML 2.0 Energy Systems module.

Time Series

The timeseries classes belong next to the schedule and weather data ones to the supporting classes. They are used throughout the other modules to represent more specific property types or additional properties.

More specifically, the timeseries classes are used to model time-varying property values, for example the *energyAmount* in *EnergyDemand*. The respective attributes have the base class *AbstractTimeSeries* as their property type.

The abstract base class has four possible specialisation classes, whereas they can be categorised into regular and irregular timeseries. Regular timeseries have a given time period (*temporalExtent*) and time interval (*timeInterval*) for the measurements. The measurement values themselves are stored in case of a *RegularTimeSeries* in a measurement list (*values*), and in case of a *RegularTimeSeriesFile* in an external file. Irregular timeseries on the other hand, provide a specific timestamp with every measurement value. They can again be modelled inline with the *IrregularTimeSeries* or stored externally with the *IrregularTimeSeriesFile* (Agugiaro et al., 2018) (Figure 17).

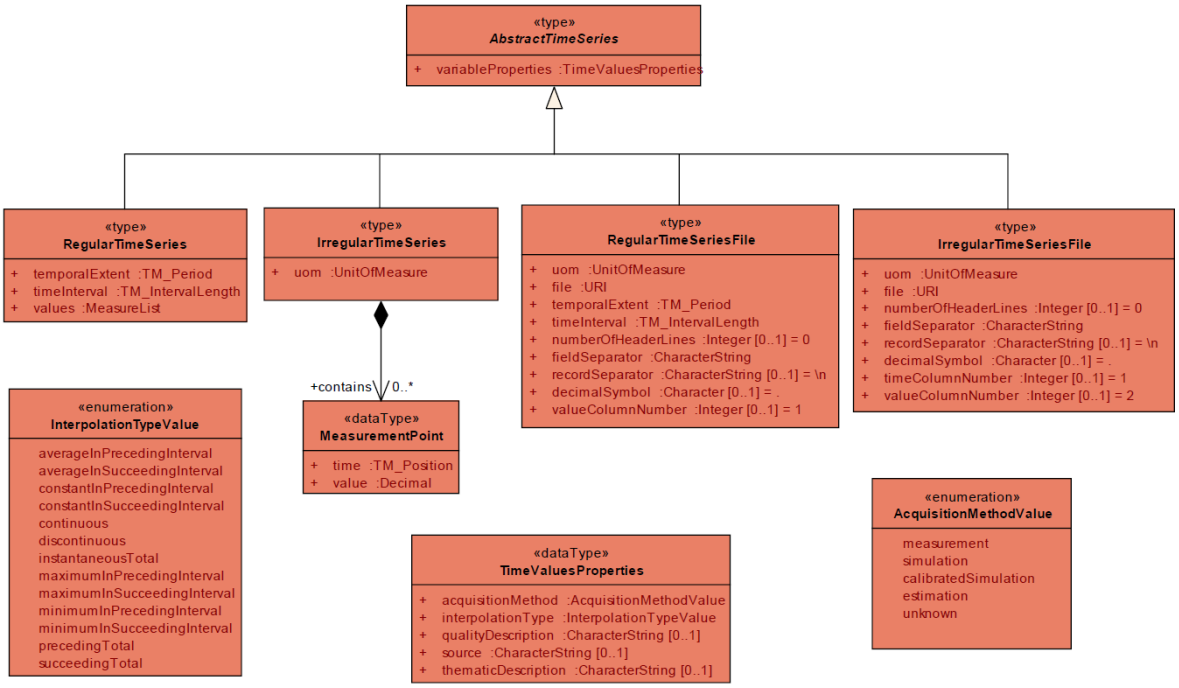


Figure 17: The Energy ADE for CityGML 2.0 Time Series supporting classes.

Schedule

Schedules are used to describe to which extent appliances or features are operated in a certain time period. They work in the same way as timeseries, by setting the property type to *AbstractSchedule* for the respective attributes.

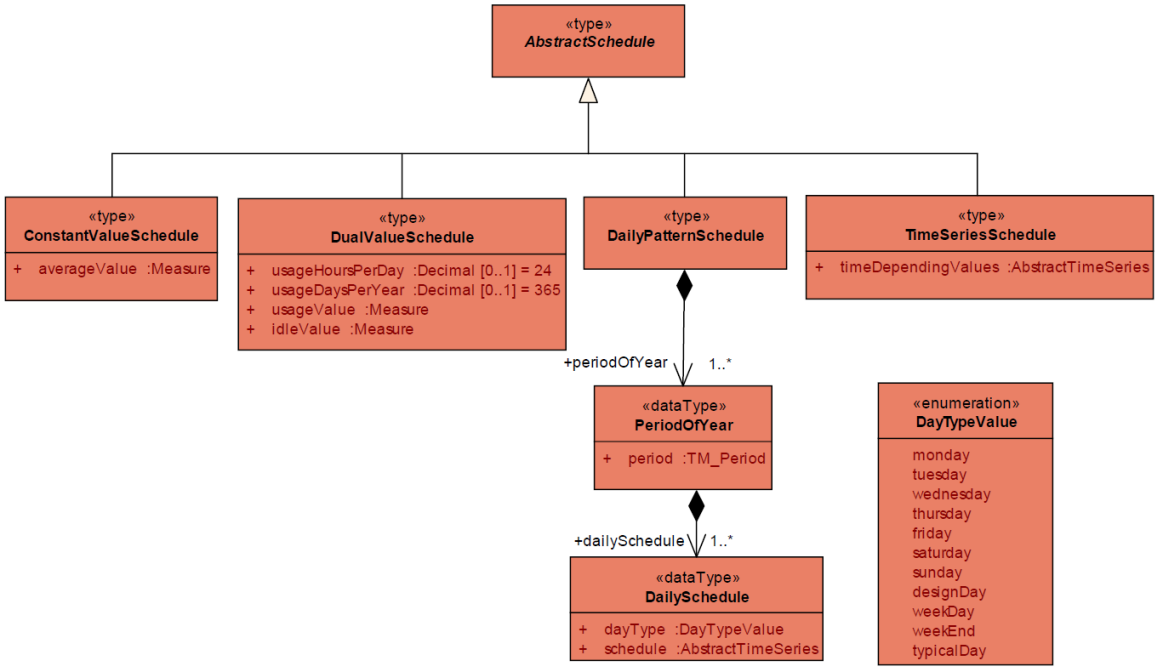


Figure 18: The Energy ADE for CityGML 2.0 Schedules supporting classes.

The base class can be specialised into four classes, with increasing degrees of freedom to model the schedules. In its simplest form, the *ConstantValueSchedule,* the usage is specified through only one average value. The *DualValueSchedule* distinguishes between operating and idle times (*usageValue*, *idleValue*). Beyond that, the *DailyPatternSchedule* allows to model varying operation times depending on the period of the year (*PeriodOfYear*) and the day (*DailySchedule*). With the *TimeSeriesSchedule* the operating times can freely be modelled through any of the given timeseries (Agugiaro et al., 2018) (Figure 18).

Weather Data

The weather data is comprised of the feature type *WeatherStation* deriving from *_CityObject*, and the type *WeatherData*, which can be added to any *_CityObject*. Therefore, the *WeatherStation* serves as a class to accumulate various time depending climate and weather metrics relevant to a more sophisticated UBEM (Agugiaro et al., 2018) (Figure 19).

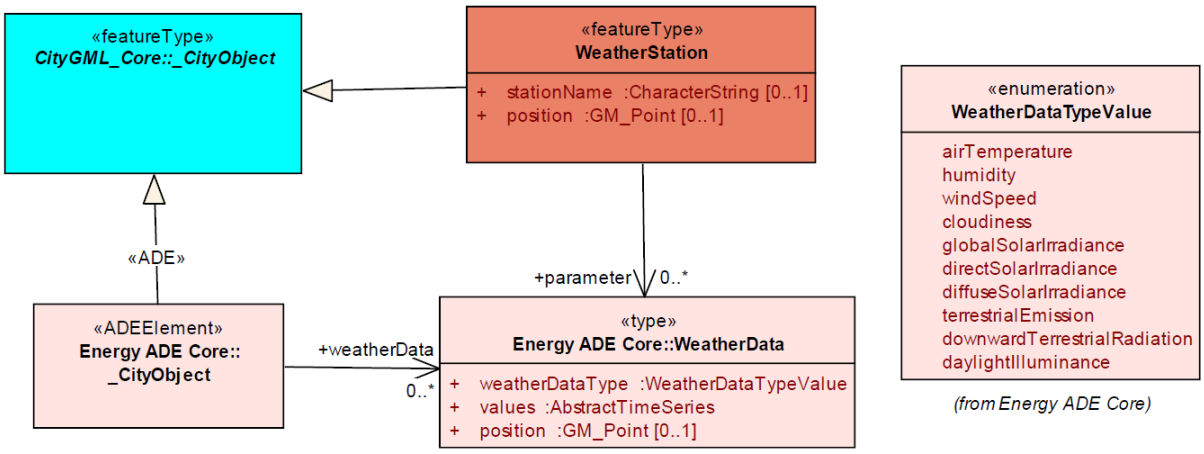


Figure 19: The Energy ADE for CityGML 2.0 Weather Data supporting classes.

KIT profile

The KIT profile is not a module of the Energy ADE, but rather a subset of it. It was developed at the Karlsruhe Institute of Technology as a simplified data model for less complex energy analysis and applications (KIT Profile, 13.11.2022). The most notable difference is the omitted Energy Systems module. Instead, the energy demand can only be modelled through the class *EnergyDemand* in the Core module. Furthermore, the supporting modules are reduced substantially. For the timeseries, only the *RegularTimeSeries* and the *RegularTimeSeriesFile* remain. The schedules are reduced to the *DailyPatternSchedule*. Beyond this, some smaller changes include the elimination of some feature types (e.g. *ReverseConstruction*), relations and properties.

2.4.3. Software Support of the Energy ADE

The vast collaboration in the Energy ADE's development likely explains the software supporting the extension. Among them is SimStadt, an urban simulation tool developed at HFT Stuttgart (Coors et al., 2021). The integrated workflow for heat demand analysis reads CityGML 2.0 data, calculates the yearly space heating and domestic hot water demands, and exports the result in form of CityGML 2.0 + Energy ADE.

Similarly, the CitySim software (CitySim, 13.11.2022) quantifies heating and cooling demands at urban scale. The simulation considers the corresponding occupant behaviour, local radiation models, thermal building properties as well as given energy and supply systems. The results can be written in form of CityGML 2.0 + Energy ADE (Kämpf & Coccolo, 2015).

Moreover, a relational database management system implementation for the Energy ADE exists. The 3DCityDB, managing CityGML data within a spatial database system, also has an extension for the mentioned KIT profile. Through this, corresponding data can be imported, analysed, manipulated and exported (3CityDB Energy ADE, 13.11.2022)

Lastly, the ETL software FME can handle the extension when provided the according XSD schema file.

2.4.4. Related ADEs

There are a few other ADEs that follow a similar purpose or share some overlap with the Energy ADE. Among them are two ADEs, both with the name Energy Efficiency ADE. They enable to model thermal zones (Dalla Costa et al., 2011) and define classes to model material properties respectively (Prieto et al., 2012). However, they are not as extensive and widespread as the Energy ADE. Furthermore, the UtilityNetwork ADE focuses on modelling utility and infrastructure networks for cities, such as electricity, fresh water, gas or telecommunication networks, and additionally provides ways to represent information about materials, usage and population (UtilityNetwork ADE, 12.11.2022) (Kutzner et al., 2018; Becker et al., 2011; Biljecki et al., 2018). As such, the ADE shows some overlap with the Energy ADE in the thematic scope but also in some of the modules that are designed. However, as opposed to the Energy ADE, the UtilityNetwork ADE focuses rather on a city level and on "what's outside the buildings" (Agugiaro et al., 2018). By this, the two ADEs complement each other and enable the user to model and simulate energy-related questions holistically for a city.

Beyond this, the Energy ADE 1.0 has also been adapted as an extension for CityJSON. The CityJSON Energy extension implements the simplified KIT profile for CityJSON 1.0 (based on the CityGML 2.0 standard) (Tufan, 2022).

2.5. CityGML 3.0

CityGML 3.0 was developed with the goal in mind to be more applicable for different user groups and additional use cases. More specifically, the interoperability with standards like IFC or INSPIRE was enhanced and additional modules were added, e.g. to incorporate the processing and storage of point clouds (Kutzner et al., 2020).

The first discussions and change requests for a new CityGML version started as early as 2013, one year after the release of CityGML 2.0. Following several years of work, the first refined

version of CityGML 3.0 in form of the GML encoding and UML diagrams was freely available in 2019 through a Git Repository (CityGML-3.0Encodings, 10.11.2022) (Kutzner et al., 2020). Eventually, the new version was officially released in September 2021, defined as a Conceptual Model standard specified through UML class diagrams. This way the new standard is encoding independent, opening the door for various implementations of the data model (Kolbe et al., 2021). The corresponding XML-based schema files are expected to be officially approved and released by the OGC in early 2023 according to Dr. Kutzner. However, they are already available through a newly set up Git Repository for the CityGML 3.0 GML encoding (CityGML3.0-GML-Encoding, 10.11.2022).

An alternative official OCG encoding for CityGML is CityJSON, developed by the 3D geoinformation research group at the Delft University of Technology. As the name already suggests, it is a JSON-based encoding of the CityGML data model, aiming to be more developer friendly through easy visualisation and manipulation of the corresponding files (Ledoux et al., 2019). Moreover, CityJSON already implements a subset of the CityGML 3.0 conceptual model since version 1.1 (CityJSON Specification, 10.11.2022). However, it deliberately does not support all features of the conceptual model standard, either due to their rare usage or to avoid an overcomplication of the encoding (CityJSON, 10.11.2022).

Additionally, a database schema encoding as it is available with 3DCityDB for CityGML 2.0 is currently in development. But there are no publications yet in this regard.

2.5.1. Changes in CityGML 3.0

With CityGML 3.0 come some major revisions. They can be summarised in four aspects. First, is the already mentioned formal definition through the Conceptual Model standard (Kolbe et al., 2021) to ensure encoding independence. This also includes its foundation on the ISO 191xx standards. Second, is the introduction of new modules (Dynamizer, Versioning, PointCloud, Construction) and the alteration of already existing ones (Core, Generics, Building, Transportation) (Figure 20). Third, the Core module now contains a new space concept and centrally defines all geometries including a revised notion of the LODs. Lastly, the ADE mechanism has been refined to easily incorporate multiple extensions at the same time and to allow for the application of the model-driven approach. Nevertheless, despite all these changes, lossless conversion of CityGML 1.0 and 2.0 datasets to CityGML 3.0 is ensured (Kutzner et al., 2020).

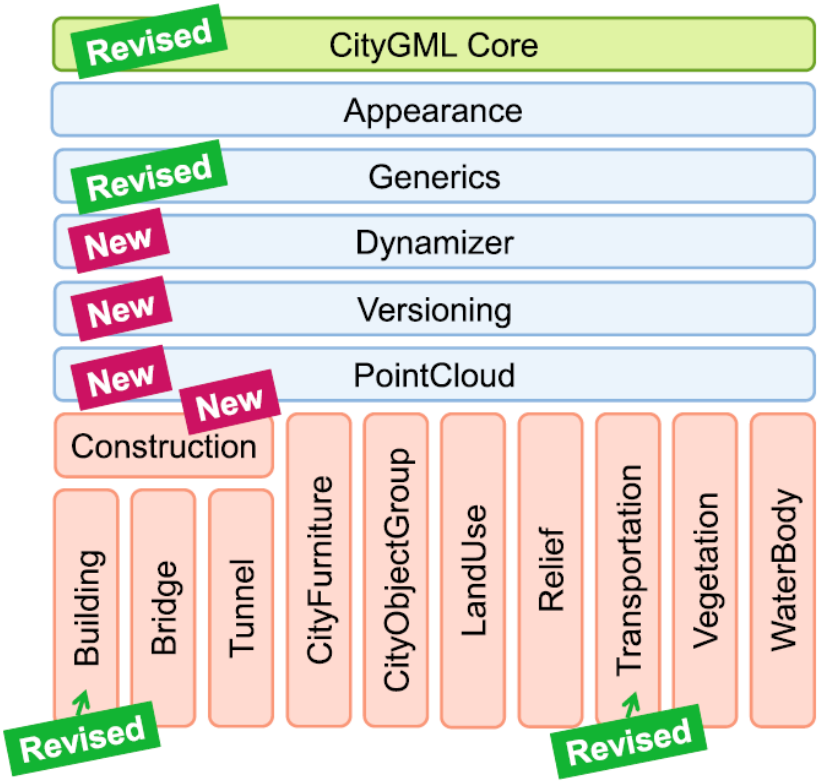


Figure 20: Overview of the CityGML 3.0 modules. Taken from Kutzner et al., 2020.

Space Concept

The newly introduced space concept is one of the major changes of CityGML 3.0. It is defined in the Core module through abstract classes. This way, another level of semantic meaning is added to each city object while inheriting the corresponding properties from the Core. An overview of the entire space concept is given in Figure 21.

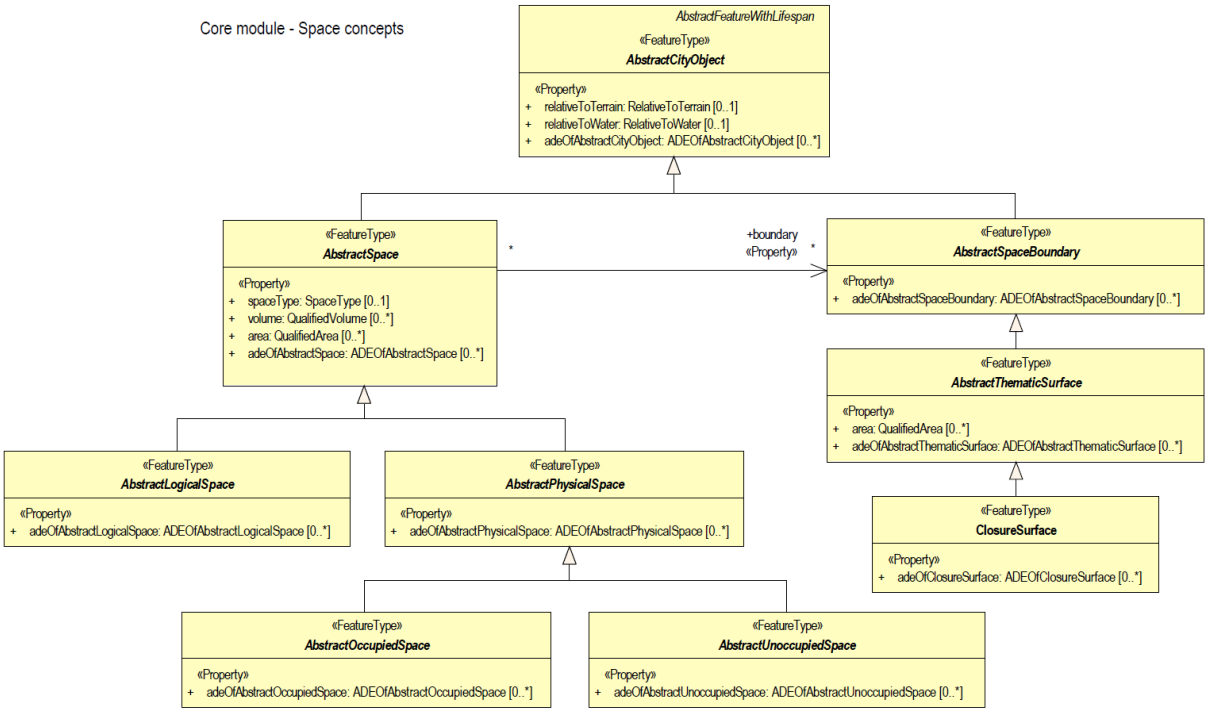


Figure 21: The space concept as defined in the Core module. Source: Kolbe et al., 2021.

On the first level, all *AbstractCityObjects* are divided into *AbstractSpace* and *AbstractSpaceBoundary*. The *space* class is used to model real-world volumetric entities, such as buildings or traffic spaces. It moreover now holds properties to further describe geometric characteristics of the objects, namely their *volume* and *area*. *Space boundaries* on the other hand describe features with an areal extent delimiting space objects. An example of this are all *AbstractConstructionSurface* elements (Construction module) such as *WallSurface* or *RoofSurface* (Figure 22).

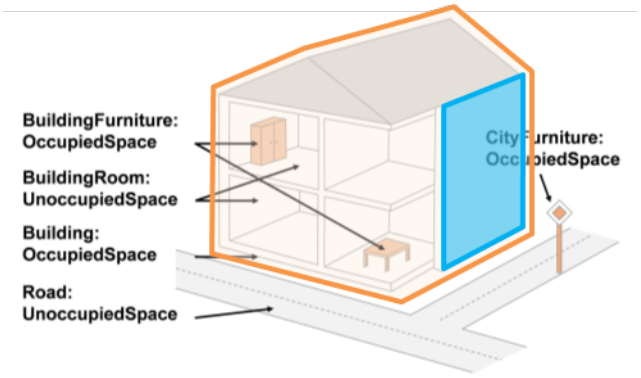


Figure 22: Space (orange) and space boundary (blue). Adapted from Kutzner et al., 2020.

In the next step, *spaces* are further subclassed into *AbstractLogicalSpace* and *AbstractPhysicalSpace*. *Physical spaces* are "fully or partially bounded by physical objects" and can hence be physically experienced (Kutzner et al., 2020). An example are buildings,

which are bounded by a roof, walls, and floor surfaces. On the other hand, *logical spaces* describe entities in regard to a thematic meaning. This can be a union of several physical spaces, such as an apartment consisting of several rooms, but also a more abstract space like a traffic zone. As such, *logical spaces* are bound by either virtual or real-world boundaries which are both geometrically modelled through *space boundaries*.

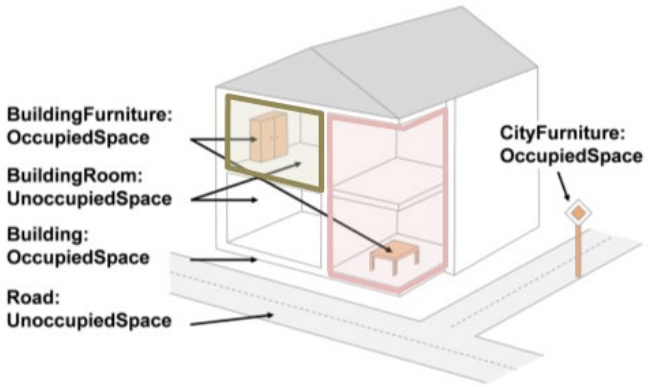


Figure 23: Physical space (room, green) and logical space (aggregation of multiple rooms, pink). Adapted from Kutzner et al., 2020.

AbstractPhysicalSpace is in turn further subclassed into *AbstractOccupiedSpace* and *AbstractUnoccupiedSpace*. *Occupied spaces* are "physical volumetric objects that occupy space in the urban environment" (Kutzner et al., 2020). It can be understood as an object which occupies space for other things to be put there. Similarly, *unoccupied spaces* are also physical volumetric objects, but they do not block any space for other entities. An example to help understand this concept is a building and its interior. The building represents an *occupied space*, as this part of the land cannot be used anymore for other purposes. However, the rooms inside describe an *unoccupied space* because it is still possible for someone to walk through or to fill it with other objects, for example, Furniture (which again would be an occupied space) (Figure 24).

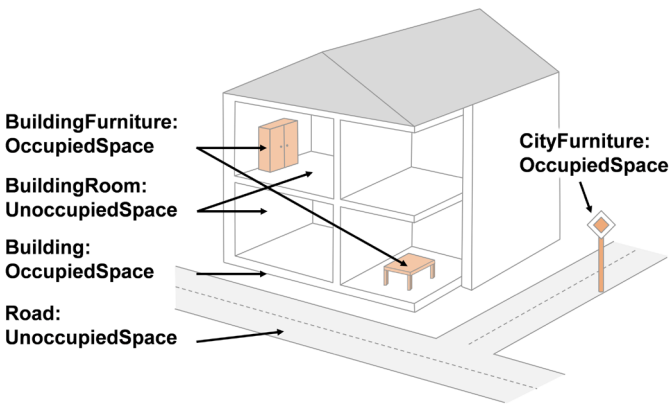


Figure 24: Occupied and unoccupied space. Taken from Kutzner et al., 2020.

Finally, the concrete classes of e.g. *Building* or *BuildingRoom* are all subclasses of these abstract space classes. Whether an object is a subclass of *space boundary*, *space* or any other of the mentioned space classes, is solely dependent on its semantic meaning (Kutzner et al., 2020).

Geometry and LOD Concept

Next to the new space concept, also all the geometries are now centrally defined in the Core module. As such, they are associated with the space concept and are thus inherited by the specialisation classes. This way, the thematic modules themselves are slimmed down, by avoiding repetitive modelling within each of them.

Moreover, the LOD concept has been revised in the new CityGML version. LOD 4, which was previously used to model the interior of buildings, has been removed. Instead, the interior can be integrated into any other LOD (0-3). Additionally, it is now possible to represent the interior itself in various levels of detail. This allows for more flexibility in terms of geometric representations depending on the specific user needs. For instance, it is possible to model a building in LOD 1 while at the same time depicting the interior in LOD 3 (Kutzner et al., 2020).

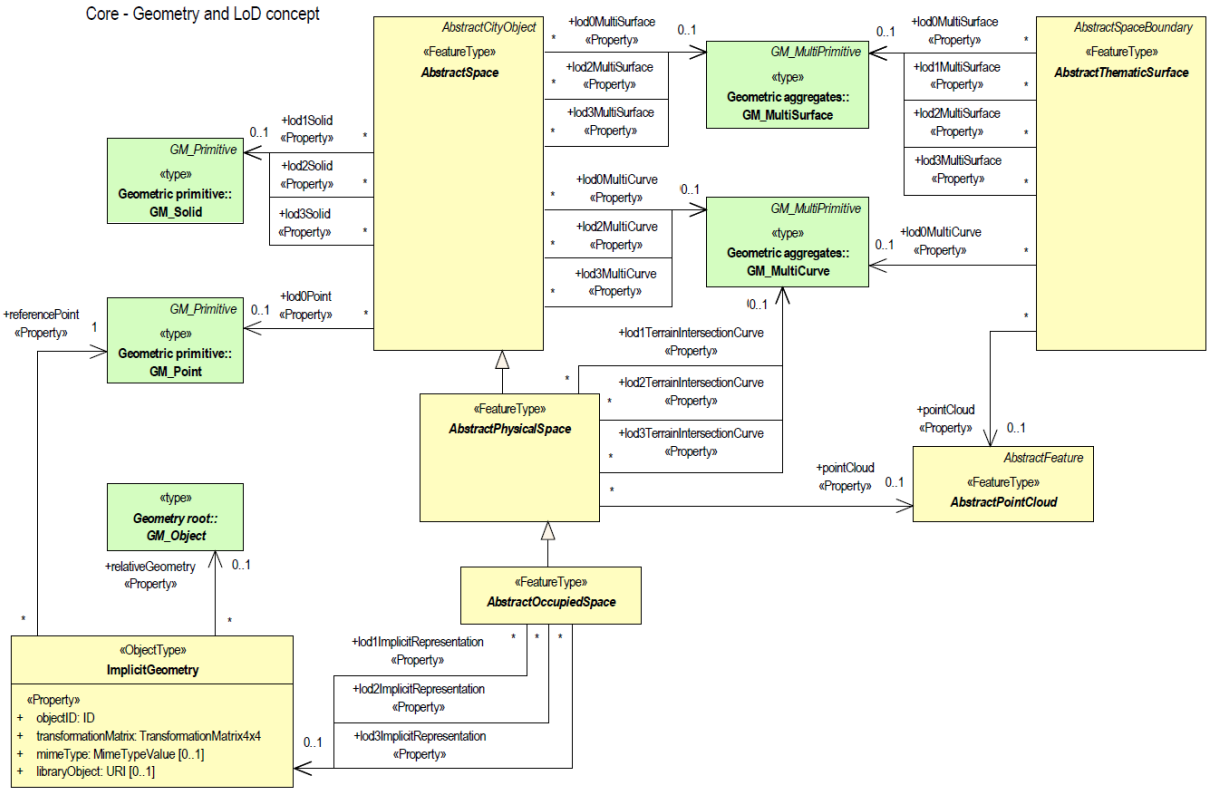


Figure 25: Excerpt of the Core module, showcasing the geometry and LOD concept. Source: Kolbe et al., 2021.

Construction module

The Construction module is newly introduced with CityGML 3.0 and serves as an intermediary module between the Core module and the Building, Bridge and Tunnel modules. It defines common classes that are used to model man-made constructions (Kutzner et al., 2020). Among them are all the thematic surfaces (*AbstractConstructionSurface, AbstractFillingSurface* and specialisation classes), the *AbstractConstruction* class which is the superclass of *Building* (Building module), and *OtherConstruction* covering other kinds of constructions. Furthermore, it contains a class *AbstractConstructiveElement* that can be used to model volumetric elements of a construction like walls or slabs (Kolbe et al., 2021).

Building module

As already mentioned, the geometries and thematic surfaces are no longer defined in the Building module. Apart from this, the *AbstractBuilding* class remains to have the two subclasses *Building* and *BuildingPart*. But now there is a composition relationship between the subclasses, with *Building* being the owning class. The earlier CityGML version has an aggregation between *AbstractBuilding* and *BuildingPart*, denoting that a *BuildingPart* can consist of further *BuildingParts*. Furthermore, the module now contains the class *AbstractBuildingSubdivision* as a subclass of *AbstractLogicalSpace* and a superclass of *BuildingUnit* and *Storey*. This enables to represent logical building subdivisions according to a homogenous property (e.g. function or ownership) in a standardised way (Kutzner et al., 2020).

Dynamizer and Versioning modules

In order to model qualitative and quantitative changes of a city or city objects over time, two new modules have been introduced, the Versioning module and the Dynamizer module.

The Versioning module allows to model qualitative variations of city objects over time. By this, it is possible to host multiple versions of the same city object within one city model as well as multiple versions of the whole city model. The Dynamizer module on the other hand, enables to represent qualitative temporal variations of attribute values which are changing frequently over time, such as the temperature throughout a day/month/year. It is based on an according ADE for CityGML 2.0 by Chaturvedi & Kolbe, 2017, which was developed to “support real-time sensor readings and other time-dependent properties within semantic 3D city models”. The

related data sources are usually either through simulations, recorded data or real-time sensors (Kutzner et al., 2020).

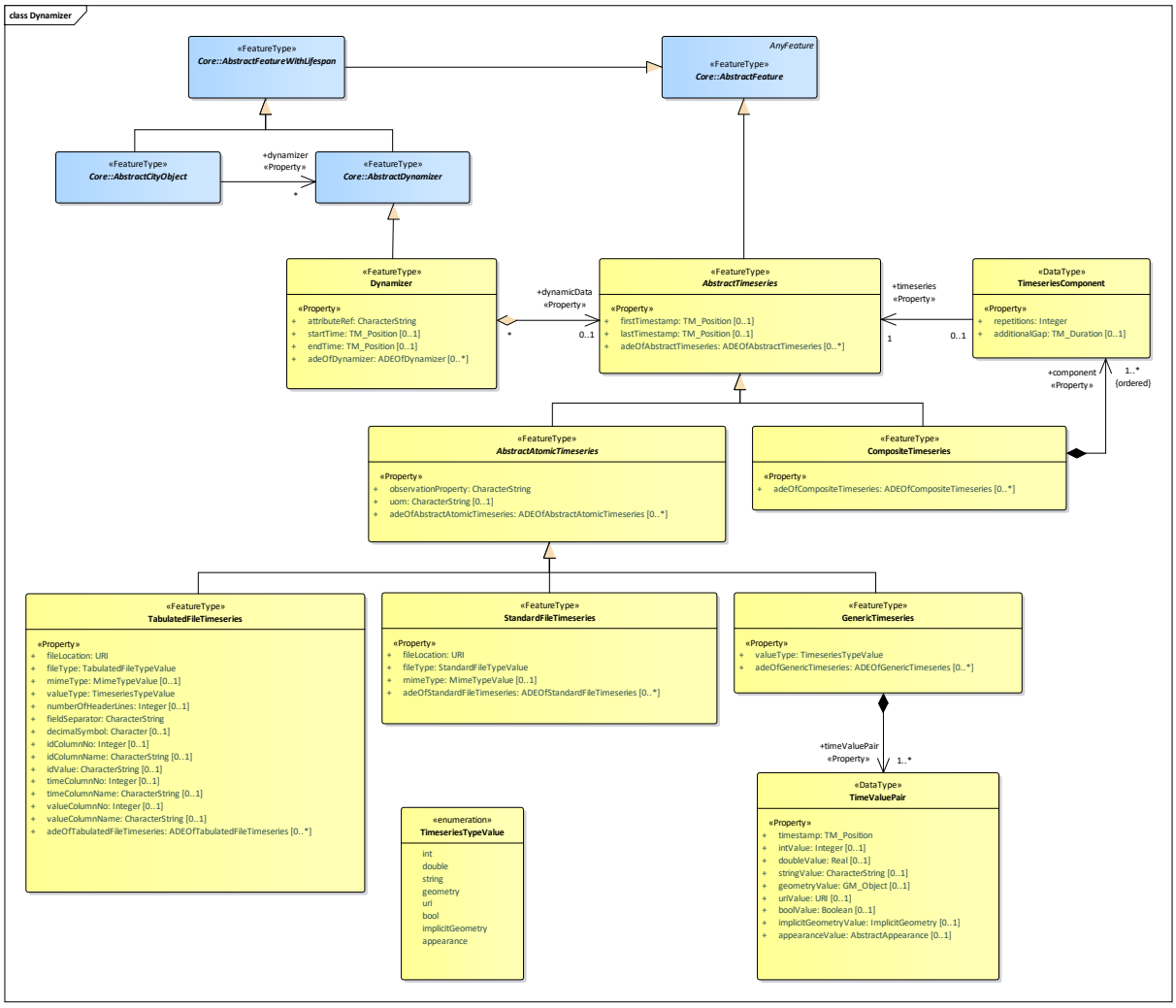


Figure 26: Excerpt of the CityGML 3.0 Dynamizer module. Source: Kolbe et al., 2021.

In the context of this thesis, the new possibility to model time-varying property values is of great importance. Thus, this aspect of the Dynamizer module is explained in more detail (see Figure 26 for an overview of the module).

At the modules core is the *Dynamizer* feature type, which can be referenced by any *AbstractCityObject* via the *dynamizer* relation. With the *Dynamizer*'s mandatory property *attributeRef*, it is indicated which exact property of which class instance is referenced and overwritten by the time-varying data. The *Dynamizer* can then own an *AbstractTimeseries*, specifying the observation property's values over time. Eventually, this can be implemented in various ways depending on the chosen specialisation class. One option is to individually model each value with a corresponding time-value-pair (*GenericTimeseries*). If the values with their timestamps are stored in an external file, they can be included via the *TabulatedFileTimeseries*

or the *StandardFileTimeseries*. For more complex timeseries, the *CompositeTimeseries* and the *TimeseriesComponent* are foreseen.

ADE mechanism

The CityGML extension mechanism has been revised in order to improve the application of several ADEs at the same time. This is achieved through the remodelled ADE hook explained below. Additionally, ADEs now have to be, just as CityGML 3.0 itself, formally defined by UML class diagrams to ensure encoding independence.

The first extension mechanism, for defining new ADE Elements, stays principally the same as in earlier versions. New features are supposed to be derived from *AbstractFeature* or an appropriate subclass of it. In case the feature has a spatial extent, it should derive from a semantically logical superclass inheriting the earlier described space concept. However, it is still possible to explicitly define geometries for new features.

The second ADE mechanism injects additional properties to already existing CityGML feature types. This form of extension is also known as ADE hook. The hook mechanism has been revised in CityGML 3.0 so that subclassing the feature type is no longer necessary. Instead, the new properties are added via the extension attribute "adeOf*FeatureTypeName*" of type "ADEOf*FeatureTypeName*" which every CityGML feature type contains. *FeatureTypeName* is replaced with the according class name (e.g. *adeOfBuilding* with the type ADEOfBuilding). The new properties are then added to a new subclass of the data type ADEOf*FeatureTypeName* with the stereotype «DataType».

Both of the mechanisms are schematically depicted in the following figure.

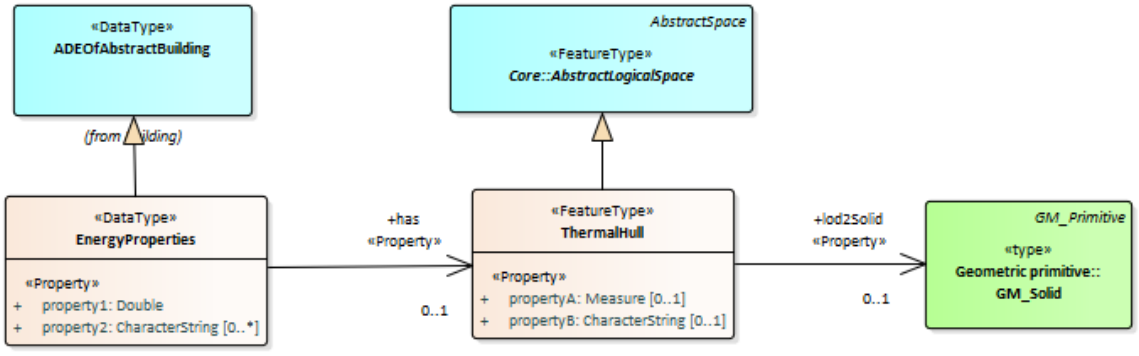


Figure 27: ADE mechanism for defining additional properties (EnergyProperties) and creating new classes (ThermalHull).

2.5.2. Applications of CityGML 3.0

Even though CityGML 3.0 is a rather new standard, it has already been used for a number of research projects and implementations. Some of these publications date back to before the release date. This is possible as the developments of the standard have always been publicly accessible through the Git repositories. However, it should still be mentioned that many of those papers are (co)written by authors who significantly contributed to the development of CityGML 3.0.

One category of published research regarding CityGML 3.0 highlights and explains the changes that are made, without an explicit application (Löwner et al., 2014), (Löwner & Gröger, 2017), (Kutzner & Kolbe, 2018), (Kutzner et al., 2020).

The other category covers actual applications and testing of the new standard. Beil et al., 2020 for instance, explore the capability of CityGML 3.0's Transportation module to model and represent streetspaces in comparison to other common formats. Another work from Beil et al., 2021 showcases how the new PointCloud module in CityGML 3.0 can be utilised to store point clouds with richer semantics than just the classification values. Beyond this, Yan et al., 2021 make use of CityGML 3.0 and IndoorGML 1.0 to develop a system for seamless indoor-outdoor navigation.

Further practical work involves the creation and usage of ADEs and is discussed in the following section.

2.5.3. ADEs for CityGML 3.0

The Git Repository for the CityGML 3.0 GML encoding contains next to the UML diagrams and schema files also example data for the different modules (CityGML3.0-GML-Encoding, 12.11.2022). Among them are also two ADEs, the Test ADE and the Urban Planning ADE.

The Test ADE is an artificial extension for CityGML 3.0 originally developed to test the support of the 3D City Database introduced with version 4.0. It covers the ADE extension mechanisms explained earlier, namely adding new feature types and extending existing ones with additional properties. The ADE consists of one UML diagram module showcasing these techniques (TestADE, 12.11.2022).

The Urban Planning ADE was developed as part of the "i-Urban Revitalization" (i-UR) project supported by the Japanese government to aid municipalities analysing and visualising

developments in regard to urban planning. The ADE enables among others to store and model information about urban zones and its functions, the population living in them, as well as the transportation accessibility (Akahoshi et al., 2020). The resulting ADE together with its UML diagram has since been partly adapted to comply with the new CityGML 3.0 standard and its revised ADE mechanism (Urban Planning ADE, 12.11.2022).

Another ADE that is implemented for CityGML 3.0 is the IfcADE by Biljecki et al., 2021. Although the new Conceptual Model aims for a better interoperability between IFC and CityGML, some relevant information is still lost. The IfcADE tries to close this gap. It is based on the preliminary Conceptual Model and XML-based encoding which were available at the time of development. Based on the ADE's corresponding UML diagram, the new extension mechanism has not been applied yet. However, according to the authors, the ADE is to be updated if necessary with the official release of the standard (Biljecki et al., 2021).

Furthermore, the already discussed UtilityNetwork ADE has been adapted to fit the new standard. This includes changes of some parent classes, considering the introduced space concept. Geometry representations are also affected by this, in a way that most of them don't need to be explicitly modelled anymore. Instead, they are inherited from the CityGML 3.0 core (citygml3-utility-network-ade, 12.11.2022).

Other ADEs for CityGML 3.0 are among others developed for the representation of a 3D cadastre in Addis Ababa (Nega & Coors, 2022) or as support for 3D underground land administration (Saeidian et al., 2022).

2.5.4. Conversion from CityGML 2.0 to CityGML 3.0

To convert CityGML data to the new version, the open-source Java library and API citygml4j can be used (citygml4j, 12.11.2022). It supports the CityGML 3.0 Conceptual Model both in its GML and JSON encoding. Beyond that, there exists a freely available FME based conversion tool (FME conversion, 12.11.2022). The tool focusses on the Building module and its necessary adjacent classes in the GML encoding. Other objects can easily be included by a user following the example of the already existing ones.

3. Method

Overall, this thesis consists of 3 steps:

1) Mapping process and creation of UML class diagrams
2) Derivation of the corresponding XSD schema file
3) Data conversion

The first two steps follow a model-driven approach. It was applied for the development of CityGML 3.0 (Kutzner et al., 2020) and is also the proposed methodology for the creation of Application Domain Extensions by van den Brink et al., 2013. However, it is slightly adapted as the Energy ADE is only mapped and not newly created and to be conformant with CityGML 3.0. The overall workflow is summarised in Figure 28.

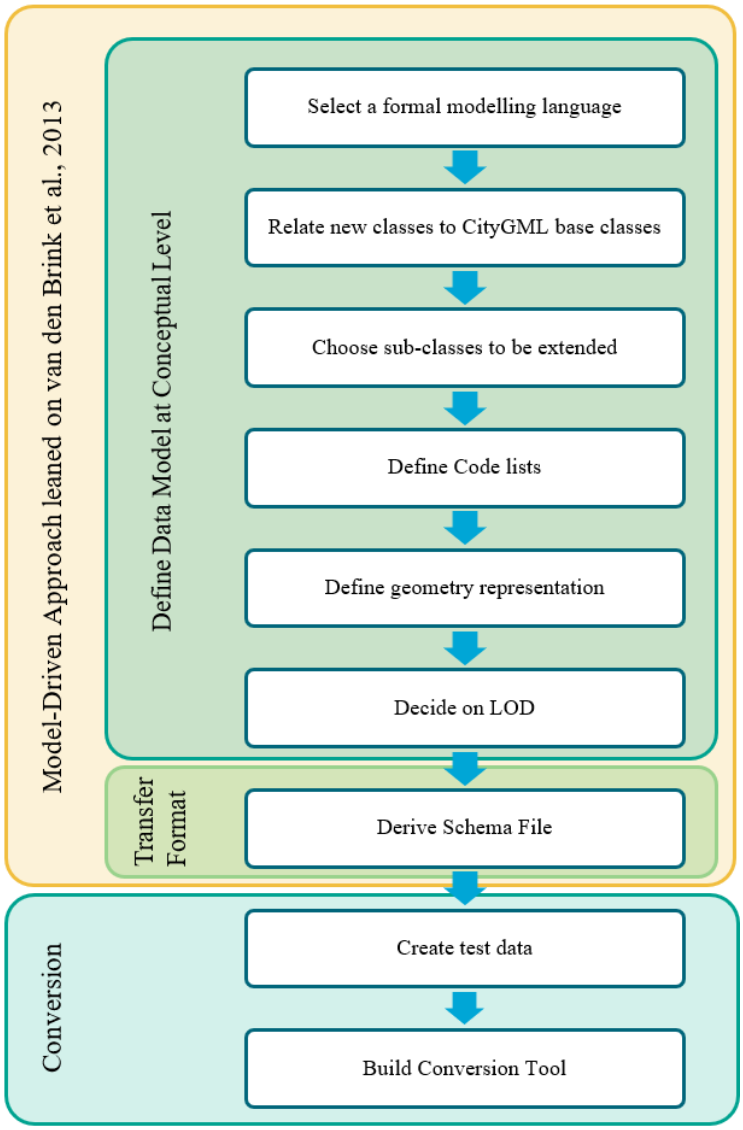


Figure 28: Schematic workflow of the mapping and conversion process.

3.1. Mapping

In the model-driven approach, first a data model defining the required classes, properties and relations is defined at a conceptual level (Kutzner et al., 2020). For this purpose, UML class diagrams are used in order to match the CityGML definition and the Energy ADE already in place. This adheres at the same time to the CityGML 3.0 conceptual model standard, requiring ADEs to be defined through UML diagrams.

To define the data model van den Brink et al., 2013 formulate six sub steps which are adapted to this research.

1) *Selection of a formal modelling language.* In this case UML is used, as it is demanded by the CityGML 3.0 specifications
2) *Establish correspondence between the CityGML base classes and the semantic classes to be included in the ADE*. This includes a first informal mapping where the new classes can potentially be placed within the CityGML UML diagram according to their semantic correspondence.
3) *Choosing of the subclasses to be extended*. Here, the specific subclass where the ADE is placed is chosen and it has to be decided which ADE mechanism applies. Either the properties will be extended through the hook mechanism, or a new class or feature will be created in relation to the existing class. This and the previous step, are already partially given through the existing Energy ADE. But due to the changes in CityGML 3.0, the correspondence and the best fitting ADE mechanism need to be revised.
4) *Possibly define code lists*. For this, the already existing codelists of Energy ADE v1.0 are mostly taken over. However, it is to be checked whether newly introduced or reworked code lists in CityGML 3.0 (partly) replace some of the Energy ADE code lists. Additionally, this step entails the assessment whether any Energy ADE properties can be replaced by newly introduced CityGML ones.
5) *Define the geometry representation for each class if necessary*. If a class in the ADE requires a geometric representation it is to be decided in which way it is needed. Through the reworked LOD and geometry concept of CityGML 3.0, most of the classes can inherit the geometries from the Core module. However, it is still possible to explicitly define the ADE classes' own geometries.
6) *Decide on topologically and semantically correct LOD*. This step comes hand in hand with the previous one. There might be cases where specific LODs are topologically

correct or incorrect. The decision should be align with the revised LOD definitions of CityGML 3.0.

This whole approach is first applied conceptually with "pen and paper", one module at a time. The general proceeding order is illustrated in Figure 29. As it can be seen the Core Module and the Supporting classes are developed in parallel to the other modules. Once the mapping is done, the according UML diagrams are created in Enterprise Architect v13 (EA). The software by Sparx Systems focuses on visual modelling of systems, software, processes and architectures (Enterprise Architect, 15.11.2022).

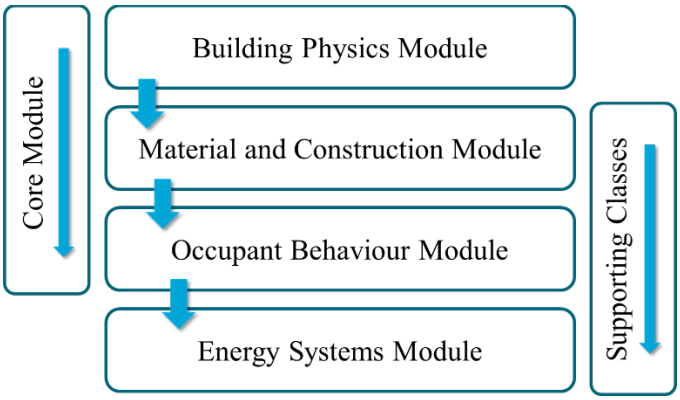


Figure 29: Proceeding order for the mapping.

3.2. XSD Schema

In the second step, the transfer format is derived from the established data model. Because this work is based on the GML encoding of CityGML, the result is in form of an XSD schema file. The file lays down the encoding rules, meaning it defines how to correctly write an Energy ADE GML file. It furthermore enables the syntactic validation of Energy ADE data.

The implementation of this step is done with the Java tool ShapeChange v2.11. In combination with a specified configuration file, the tool is able to connect to EA and automatically derives the corresponding XML schema (ShapeChange, 15.11.2022). Within this process, modelling errors and violations against the ISO19109 standard in the UML diagrams are detected as a side effect. Additionally, the resulting file is manually examined on any potential remaining problems.

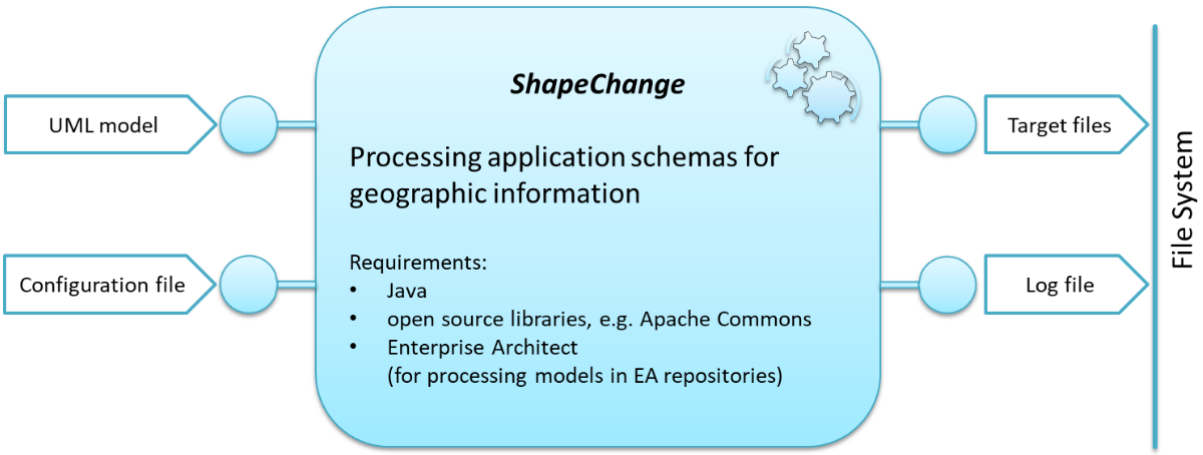


Figure 30: Overview of ShapeChange process. Taken from ShapeChange, 26.11.2022.

3.3. Conversion

The last step converts CityGML 2.0 data extended with the Energy ADE to CityGML 3.0 plus Energy ADE. In order to create and test a lossless conversion application, a test dataset needs to be generated first. This dataset aims to cover as much of the Energy ADE as possible, comprising every feature type, data type, property and relation at least once. However, there are almost endless possibilities to model them in terms of XLinks, parent classes or relations to one another. Therefore, the test dataset does not implement the Energy ADE in all its entity.

The test dataset creation, as well as the development of the conversion tool is done with the ETL software FME Desktop v2022.0. It provides means to import and export CityGML data in different versions and in combination with ADEs. Furthermore, the files can be validated against the given schemas. The conversion tool itself builds up on the described FME workspace in part 2.5.4 since it also needs to convert the relevant features of CityGML 3.0.

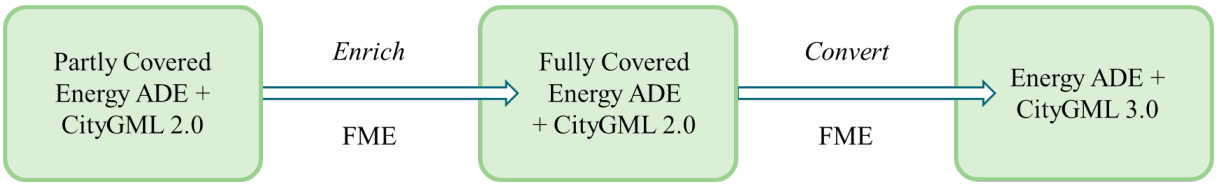


Figure 31: Test data creation and conversion.

As the starting point for the input data serves a CityGML 2.0 plus Energy ADE file which already contains several prominent features and properties of the extension. It is part of the

GEO5014 course materials at TU Delft and provided by Giorgio Agugiaro. Missing features in the dataset are identified and subsequently complemented. The such acquired data is then in turn employed as input for the conversion (Figure 31).

Finally, the converted results are verified in two ways. The first one is by the GML Writer validation, checking whether everything is in accordance with the provided XSD schema file. The second one is a manual validation of the results by comparing the input gml file with the produced output file. Through this, missing properties or wrong assignments to parent elements can be detected.

Although the implementation of this thesis overall follows the described workflow, it is not always a linear process. Certain feedback loops exist, where smaller problems or inconsistencies in previous steps are detected. These are then corrected if necessary.

4. Implementation

As described in the Methodology, the implementation consists of three steps. The structure of this chapter follows those steps, starting with the conceptual mapping of the Energy ADE 1.0 to CityGML 3.0. Next, the derivation of the XSD schema through ShapeChange is explained. Finally, a description of the two FME workspaces to create the test data and to subsequently convert it is provided.

4.1. Mapping the Energy ADE to CityGML 3.0

At the thesis' core is the mapping process of adjusting the Energy ADE to fit the new CityGML 3.0 standard. The goal is to obtain an updated extension without any changes of content and loss of information.

At the same time, the result should demonstrate coherent mapping and logical consistency throughout all the modules. In order to achieve this, two general mapping principles are established which function as a guideline, as there usually is more than just one possibility to implement something. Moreover, some overarching mapping decisions are explained. They constitute decisions which generally apply within all ADE modules.

The remainder of chapter 4.1 then demonstrates and reasons the final mapping decisions as well considered alternatives module by module.

General mapping principles

1. Integrate as much as possible:

   The first guiding principle is to integrate the Energy ADE as much as possible into the CityGML 3.0 conceptual model. This also implies to make use of the newly introduced space and geometry concept. An advantage of this strategy is that an additional layer of semantic meaning is added to each class deriving directly or indirectly from *AbstractCityObject*. It also leads to the possibility of replacing certain Energy ADE properties by newly added CityGML 3.0 ones. Furthermore, this strategy enables a multiple LOD representation and makes the explicit definition of geometries redundant.

   Alternatively, the Energy ADE classes could be kept closer together on a higher level while explicitly defining the geometries and all the individual properties. As such, the changes in

CityGML 3.0 would largely be disregarded and the final result would resemble a mere copying of the Energy ADE already in place.

2. Maintain a logical symmetry:
   The second modelling principle aims to retain a certain logical symmetry while mapping. In other words, similar classes in terms of their meaning or conceptual level should also be mapped to the same parent class or the same hierarchy level in the CityGML 3.0 data model.

On a more individual level, the decision on how and where an Energy ADE class is integrated within CityGML 3.0 depends on multiple factors. First and foremost is the semantic fit of a class to its potential parent class. It is possible that an ADE class fits several CityGML classes throughout their specialisation relations (e.g. *AbstractBuilding* or *Building*). In such cases, it is inspected whether the additional properties and relations of the more specialised class add any value to the ADE class. Moreover, it is compared how similar classes are mapped to fulfil the second modelling principle. Lastly, it is checked if the decision would inadvertently effect any of the other ADE classes (e.g. by adding properties to a class via the hook which are then in turn inherited by another ADE class).

Although the mapping principles might sound rather abstract at this point, they will become tangible through the explanations in the particular modules (see chapter 4.1). It is furthermore important to mention, that those guidelines still give room for multiple solutions in certain cases. If so, the decision is made on the individual level.

Overarching mapping decisions

1. *AbstractFeatureWithLifespan* over *AbstractFeature:*
   AbstractFeatureWithLifespan is always preferred over *AbstractFeature* as generalisation class. This way, useful additional properties can be included. Among them are the newly introduced ones *validFrom* and *validTo*. By including them it is possible for every Energy ADE object to be represented in different versions throughout its history. As such they integrate in the CityGML 3.0 way of supporting feature history.

2. Maintain abstract classes:
   A benefit of using abstract classes is to support the modular structure of UML diagrams with several packages. Through them, they can be conveniently connected with each other. Although, the in this thesis proposed mapping does not make use of this modular structure

and rather implements everything in one package, the abstract classes are kept. This follows the idea of a lossless mapping and opens the possibility for an easy transferral into the modular structure in potential future work.

3. Multiplicities, relations and properties:
   Finally, multiplicities, relations and properties are generally kept as they are in the Energy ADE for CityGML 3.0. In case something is changed in this regard, it is explicitly mentioned and reasoned.

4.1.1. Core module

BuildingProperties

The additional properties for *Building* and *BuildingPart* keep being added to the class *AbstractBuilding*. With the revised ADE hook mechanism, the subclassed data type needs a unique descriptive name. Here, *BuildingProperties* is chosen.

Several ADE properties can be replaced by CityGML 3.0 ones, namely *volume*, *floorArea* and *heightAboveGround*. The detailed mapping is demonstrated in the tables 1, 2 and 3. Figure 32 additionally shows the mapping in terms of UML diagrams on the example of *volume*. The mapping of the other two properties is alike. Furthermore, the point geometry *referencePoint* is replaced by the *lod0Point* geometry deriving from *AbstractSpace*.

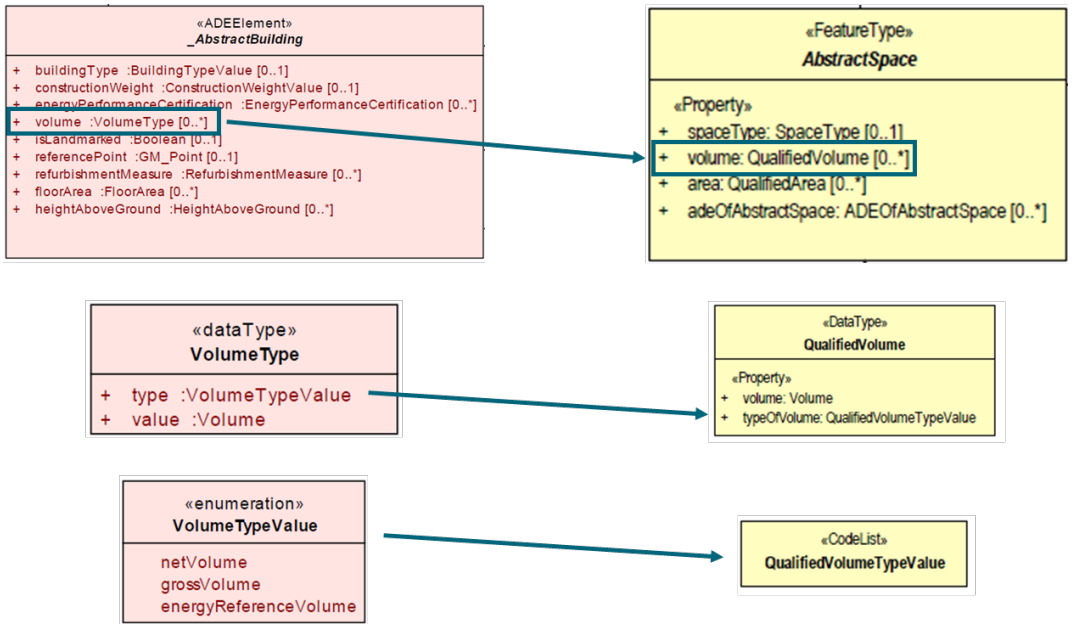


Figure 32: Mapping volume in _AbstractBuilding to volume in AbstractSpace in terms of UML. Source: Agugiaro et al., 2018; Kolbe et al., 2021.

Table 1: Integration of the volume property in Energy ADE for CityGML 2.0 into the volume property of AbstractSpace in CityGML 3.0. VolumeTypeValue is an enumeration, QualifiedVolumeTypeValue an extendible codelist.

«ADEElement » ***_AbstractBuilding***		***AbstractSpace***
volume: VolumeType		*volume*: QualifiedVolume
type: VolumeTypeValue	→	*typeOfVolume*: QualifiedVolumeTypeValue
value: Volume		*volume*: Volume

Table 2: Integration of the floorArea property in Energy ADE for CityGML 2.0 into the area property of AbstractSpace in CityGML 3.0. FloorAreaTypeValue is an enumeration, QualifiedAreaTypeValue an extendible codelist.

«ADEElement » ***_AbstractBuilding***		***AbstractSpace***
floorArea: FloorArea		*area*: QualifiedArea
type: FloorAreaTypeValue	→	*typeOfArea*: QualifiedAreaTypeValue
value: Area		*area*: Area

Table 3: Integration of the heightAboveGround property in Energy ADE for CityGML 2.0 into the height property of AbstractConstruction in CityGML 3.0. ElevationReferenceValue is a codelist and can be extended. lowReference and status don't have an equivalent in the Energy ADE. For lowReference "ground" can be set, HeightStatusValue is an enumeration of the values "estimated" and "measured".

«ADEElement » ***_AbstractBuilding***		***AbstractConstruction***
heightAboveGround: HeightAboveGround		*height*: Height
heightReference: ElevationReferenceValue	→	*highReference*: ElevationReferenceValue
value: Length		*value*: Length
		lowReference: ElevationReferenceValue
		status: HeightStatusValue

EnergyDemand

In a similar manner to the Energy ADE for CityGML 2.0, every CityObject *demands* zero to many *EnergyDemand* instances. But because association relations are in principle just properties by reference, they are not allowed to be added directly to a CityGML feature type, as this would alter the original data model. Therefore, the class in question is extended with the

ADE hook, from which the relation to the ADE object departs. In this case, the subclassed DataType is named *EnergyADECityObjectProperties*.

Within the *EnergyDemand* class itself, the compulsory property *energyAmount* is described through a timeseries. In this version of the Energy ADE, the supporting classes of *AbstractTimeSeries* are partly replaced by classes of the CityGML 3.0 Dynamizer module. Where this is not sufficient to convey the Energy ADE timeseries information, the ADE mechanisms are applied to extend the Dynamizer module. Therefore, the way time-varying properties are modelled has changed. While the detailed functioning is explained in subchapter 4.1.6, it is at this point important to know that a class containing a time-depending property requires a relation to the *AbstractDynamizer*. Such a connection is already provided by CityGML 3.0 for any *AbstractCityObject* (see Core module: *AbstractCityObject* to *AbstractDynamizer*). However, *EnergyDemand* is derived from *AbstractFeatureWithLifespan* and thus, the relation needs to be explicitly added. This is implemented by a navigable association from *EnergyDemand* to *AbstractDynamizer*. The role name (*dynamizer*) is adopted from CityGML 3.0. The multiplicity on the other hand is set to 1, signifying it is a mandatory relation. Because *energyAmount* is a mandatory property of *EnergyDemand*, and it needs to be referenced by a *Dynamizer* for the time-depending values, the multiplicity can be regarded as a "security check".

WeatherData

WeatherData objects are altered from having the stereotype «type» to «FeatureType». The former is not used for application schemas anymore, but rather for conceptual schemas (*ISO 19109:2015*, 2015). In order to preserve the ability of carrying a unique id, «FeatureType» is used instead.

As a new feature type, a suitable generalisation class is needed. Without any further specification, it would simply be a subclass of *Feature*. *WeatherData* objects carry information regarding different weather phenomena. Although they can be located with a point geometry (position), they are not a physical object in a city. Hence, *AbstractCityObject* is excluded as superclass and *AbstractFeatureWithLifespan* is chosen instead.

Beyond this, *WeatherData* shares some common characteristics with *EnergyDemand*. Every *AbstractCityObject* can also be enriched with weather information. This is again implemented with a relation from the hook data type *EnergyADECityObjectProperties* to *WeatherData* (role

name: *weatherData*). Moreover, *WeatherData* also carries a time-varying property (*values*) which requires a connection to the *AbstractDynamizer*. The respective relation is modelled in the same way.

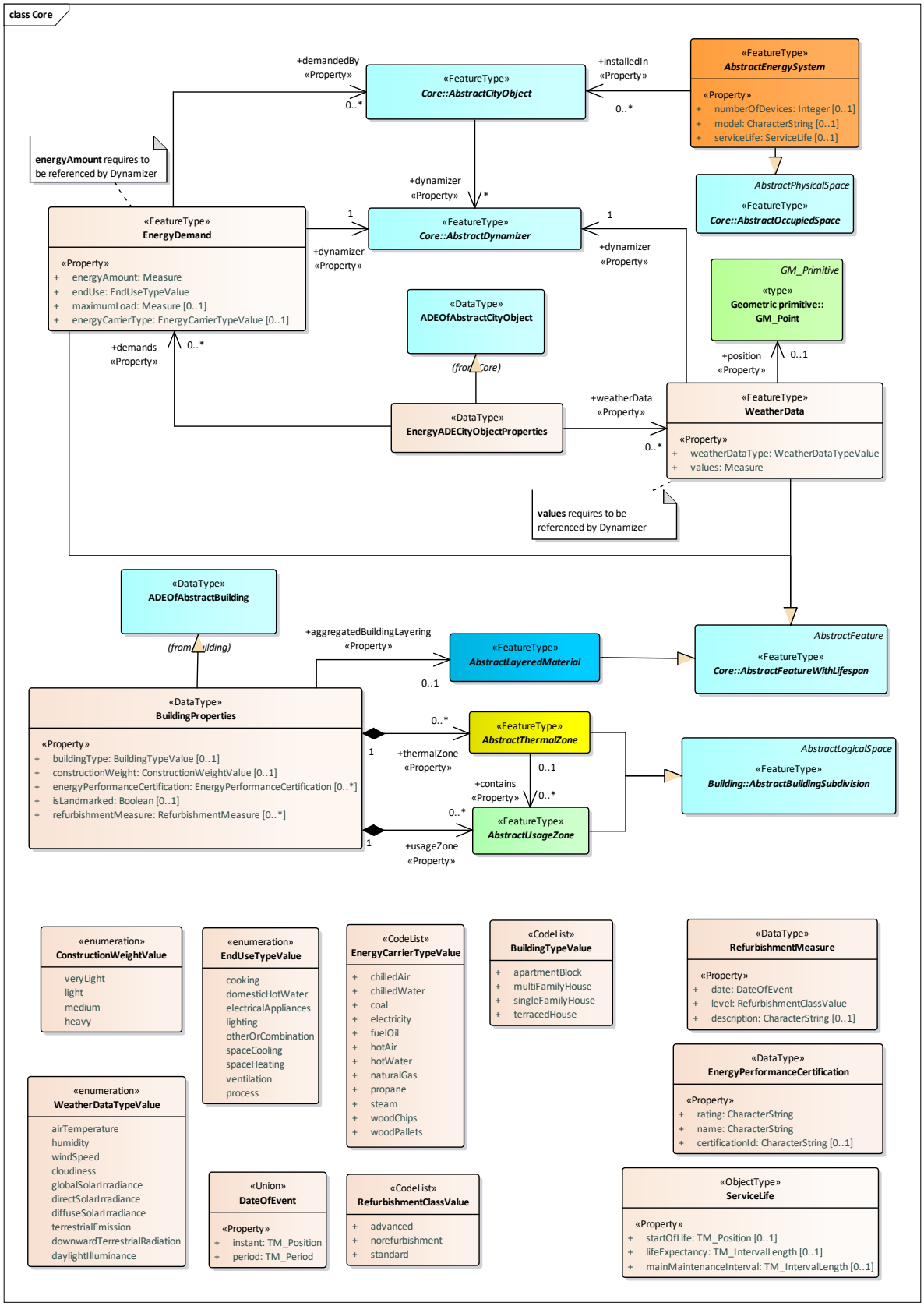


Figure 33: The Energy ADE for CityGML 3.0 Core module.

Apart from this, the properties of *WeatherData* remain. Only the inline property *position* with the point geometry is now expressed through an association relation to be aligned with the modelling style of CityGML 3.0.

4.1.2. Building Physics module

The Building Physics module showcases the general mapping principles based on the various implementation possibilities as described in the following.

One option to map the *ThermalZone*, *ThermalBoundary* and *ThermalOpening* to fit CityGML 3.0, is by deriving them all from *AbstractCityObject* (Figure 34). This is very similar to the implementation of the Energy ADE for CityGML 2.0. All properties remain in place. The required geometries need to be explicitly modelled.

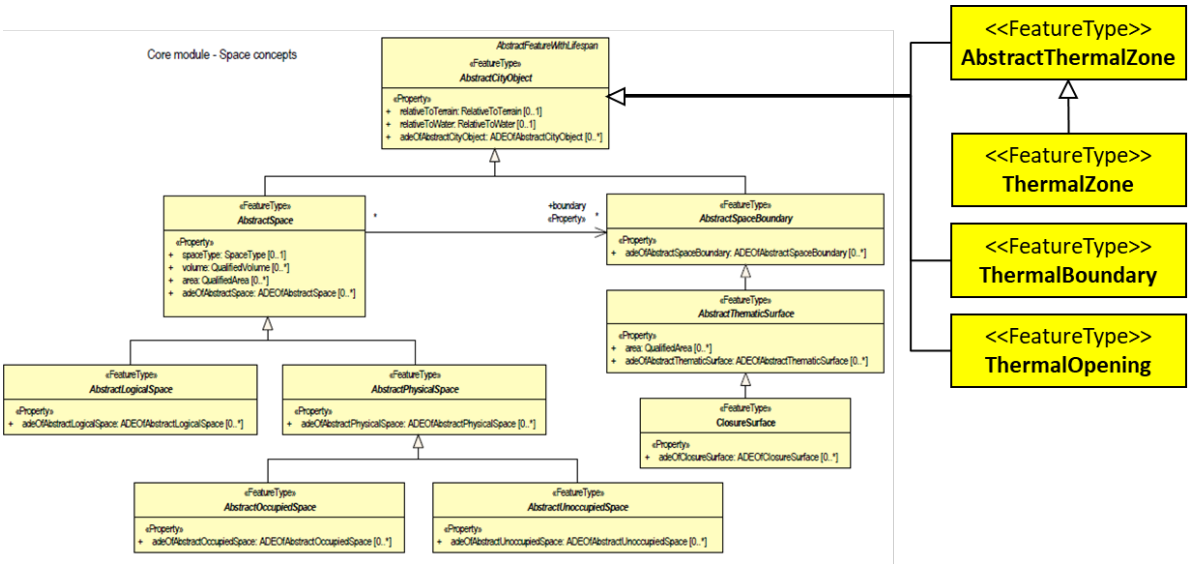


Figure 34: Option of mapping the Building Physics module to AbstractCityObject. Adapted from Kolbe et al., 2021.

The alternative is to spread the Building Physics classes across the CityGML 3.0 UML diagrams, according to their best semantic fit. For the *ThermalZone* this is some specialisation class of *AbstractSpace*, for the *ThermalBoundary* and *ThermalOpening* a subclass of *AbstractSpaceBoundary*. As such, properties can be mapped to CityGML 3.0 ones, geometries do not need to be explicitly modelled and another layer of semantic meaning is added through the space concept.

Based on these arguments, and as already anticipated, the second option is further pursued. In this scope, different mapping possibilities are debated.

(Abstract)ThermalZone

A thermal zone is an intrinsically logical concept. Therefore, it can be mapped to *AbstractLogicalSpace* or one of its subclasses (see Figure 35 and Table 4). *AbstractLogicalSpace* itself is a suitable superclass for *AbstractThermalZone*. Yet it is rather generic and according to the mapping principles a more specialised class adds additional value.

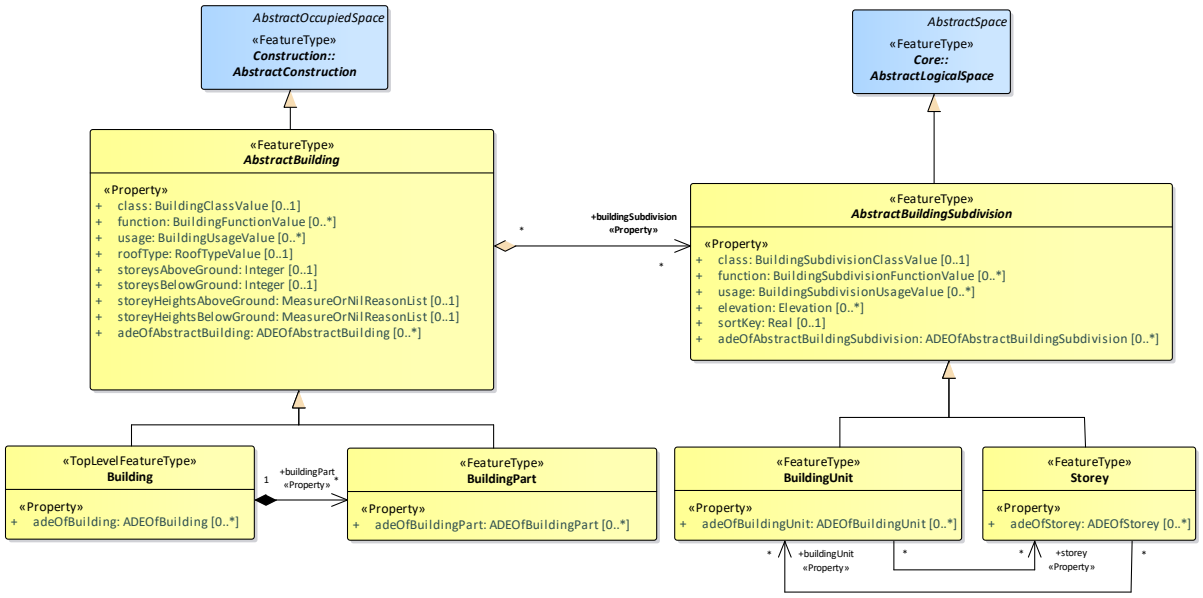


Figure 35: Excerpt of the CityGML 3.0 Building module. Adapted from Kolbe et al., 2021.

Following this logic, *BuildingUnit* constitutes a well-fitting superclass. The homogenous property defining the subdivision would correspond to the isothermal volume of a thermal zone. However, choosing *BuildingUnit* as superclass for *AbstractThermalZone* leads to interrelation issues with The Energy ADE *BuildingUnit* of the Occupant Behaviour module. Anticipating some of the mapping decisions within this module, the Energy ADE *BuildingUnit's* properties are incorporated to the CityGML *BuildingUnit* class by the ADE hook mechanism. Consequently, all those additional properties could be passed on to the *ThermalZone* leading to logical inconsistencies. Hence, *AbstractBuildingSubdivision* is selected as the parent class for *AbstractThermalZone*.

Table 4: Selected CityGML 3.0 class descriptions. Taken from Kolbe et al., 2021.

Class	Description
AbstractLogicalSpace	*AbstractLogicalSpace* is the abstract superclass for all types of logical spaces. Logical space refers to spaces that are not bounded by physical surfaces but are defined according to thematic considerations.
AbstractBuildingSubdivision	*AbstractBuildingSubdivision* is the abstract superclass for different kinds of logical building subdivisions.
BuildingUnit	A *BuildingUnit* is a logical subdivision of a Building. *BuildingUnits* are formed according to some homogeneous property like function, ownership, management, or accessibility. They may be separately sold, rented out, inherited, managed, etc.

Using the *AbstractBuildingSubdivision* as superclass, the properties *floorArea* and *volume* can be replaced by the *area* and *volume* property of *AbstractSpace* in the same manner as in the *BuildingProperties*. Similarly, the solid geometry *volumeGeometry* is replaced by the respective geometry in the CityGML 3.0 Core module.

As it can be seen in Figure 35, there already exists a relation (*buildingSubdivision*) from *AbstractBuilding*, which is extended with the *BuildingProperties* in the ADE Core, to *AbstractBuildingSubdivision* of which *AbstractThermalZone* is a subclass. Thus, it is not strictly necessary to add another relation in the ADE Core module between *BuildingProperties* and *AbstractThermalZone*. However, it increases the readability of the UML diagram to explicitly demonstrate how the objects are connected. Furthermore, the additional relation (*thermalZone*) is specified to be a composition, showcasing the stronger dependency. Its addition does not violate the CityGML 3.0 *buildingSubdivision* aggregation relation since it can simply be regarded as a subset.

The same accounts for the *boundary* relation from *ThermalZone* to *ThermalBoundary*. Although a *boundary* relation is already provided by CityGML 3.0 between *AbstractSpace* and *AbstractSpaceBoundary*, of which *ThermalZone* and *ThermalBoundary* are subclasses, it is again modelled and further restricted in the Building Physics module. Additionally, the name of the relation is changed from *boundedBy* in the Energy ADE for CityGML 2.0 to *boundary* in order to match the wording in CityGML 3.0.

Lastly, the target class of the association *interiorRoom* originating from *ThermalZone* is updated from *Room* in CityGML 2.0 to *BuildingRoom* in the CityGML 3.0 Building module.

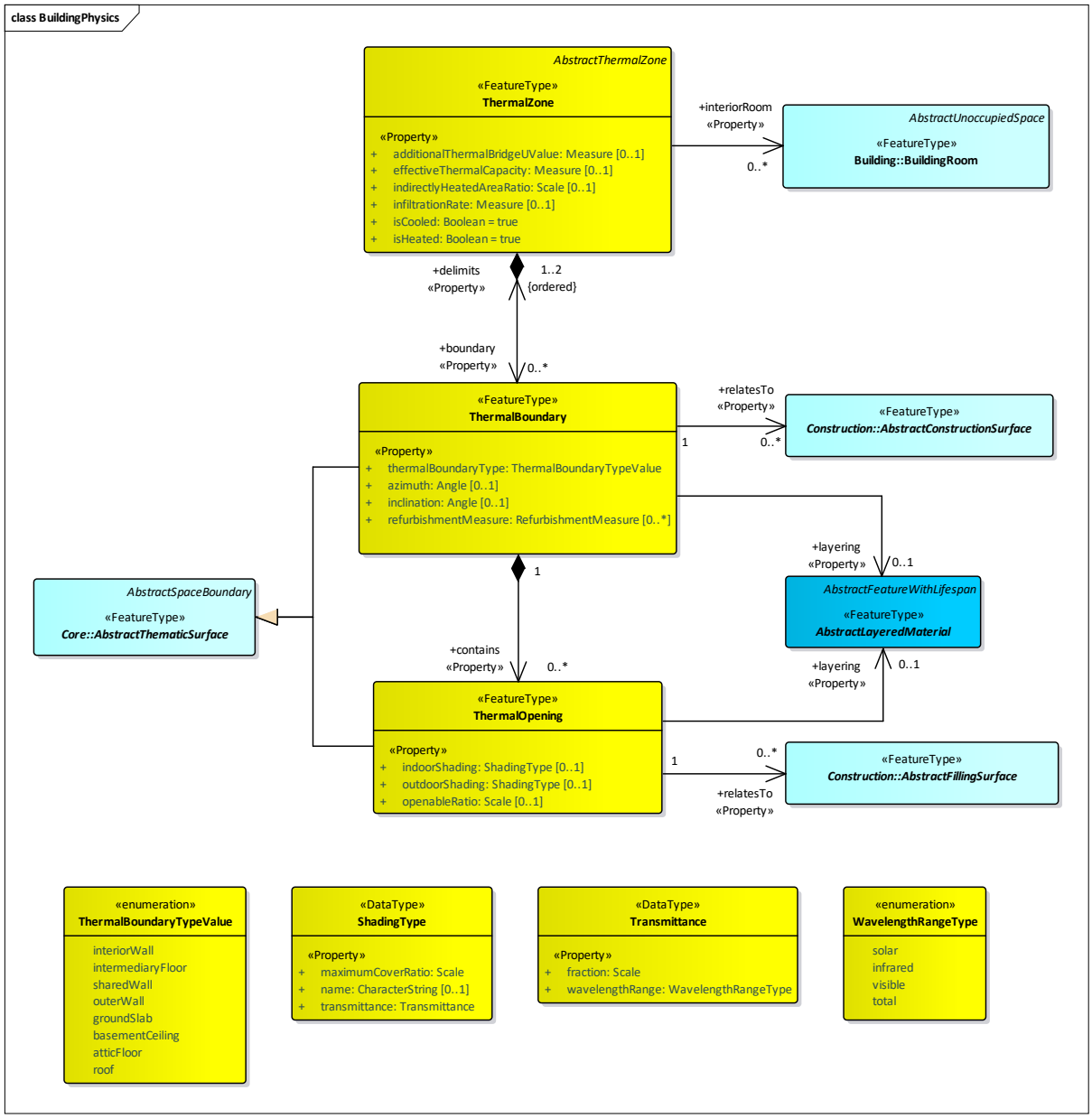


Figure 36: The Energy ADE for CityGML 3.0 Building Physics module.

ThermalBoundary and ThermalOpening

ThermalZones are fully enclosed by *ThermalBoundaries* and *ThermalOpenings*. They both have an areal extent, thus it is evident that they are mapped to a subclass of *AbstractSpaceBoundary*.

In order to make use of additional properties and the CityGML 3.0 geometries, the highest possible parent class is *AbstractThematicSurface* (see Figure 25).

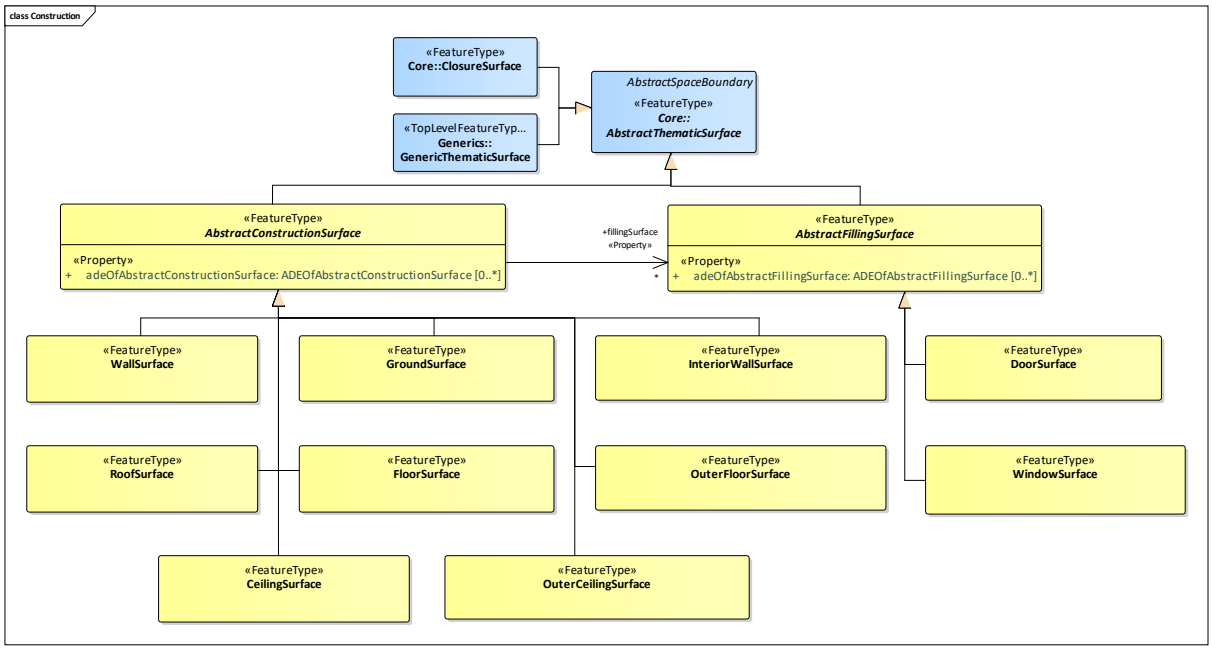


Figure 37: Excerpt of the CityGML 3.0 Construction module. Adapted from Kolbe et al., 2021.

The class *ClosureSurface* was briefly considered as parent class for *ThermalOpening*, but looking at its description (see Table 5) reveals a semantic mismatch. *ClosureSurfaces* are rather used to fill holes in volumetric objects (e.g. the openings of a tunnel), whereas *ThermalOpenings* describe thermal discontinuities in a surface, namely windows and doors.

Table 5: Selected CityGML 3.0 class descriptions. Taken from Kolbe et al., 2021.

Class	Description
AbstractThematicSurface	*AbstractThematicSurface* is the abstract superclass for all types of thematic surfaces.
ClosureSurface	*ClosureSurface* is a special type of thematic surface used to close holes in volumetric objects. Closure surfaces are virtual (non-physical) surfaces.
AbstractConstructionSurface	*AbstractConstructionSurface* is the abstract superclass for different kinds of surfaces that bound a construction.
AbstractFillingSurface	*AbstractFillingSurface* is the abstract superclass for different kinds of surfaces that seal openings filled by filling elements.

Another possibility is to map *ThermalBoundary* to *AbstractConstructionSurface* and *ThermalOpening* to *AbstractFillingSurface*. This has the advantage of the already established association relationship between the two classes (*fillingSurface*, see Figure 37). Moreover, the concept of a thermal opening perfectly fits the *AbstractFillingSurface*. In most cases thermal openings occur at windows or doors and *AbstractFillingSurface* is specifically made to model these surfaces. On the other hand, the *ThermalBoundary* class does not fit the

AbstractConstructionSurface ideally. The CityGML 3.0 class refers to surfaces bounding a construction (referring to the CityGML 3.0 concept of a construction, see Table 6). However, a *ThermalZone* is a logical concept and a *ThermalBoundary* is consequently not a construction surface of physical nature.

Now, it is possible to model *ThermalBoundary* and *ThermalOpening* to different levels within the CityGML UML diagram. But one of the goals while mapping is to retain a sort of logical symmetry, meaning similar classes should be derived from the same or comparable parent classes. This ensures consistency and thus an easier understanding of the Energy ADE UML class diagrams for a user.

Therefore, in the case of *ThermalBoundary* and *ThermalOpening* a compromise has to be made. Either derive them both from *AbstractThematicSurface* or use *AbstractConstructionSurface* and *AbstractFillingSurface* instead. While both scenarios are legitimate, the decision is made to uniformly use *AbstractThematicSurface* as superclass. Eventually, using a more generic parent class for *ThermalOpening* does not make it less correct. It only comes at the cost of losing some additional semantic context. Whereas, mapping a class somewhere with a slight semantic mismatch contravenes the established mapping principles of CityGML 3.0 (stating the determining factor for mapping is the semantic fit).

For both, *ThermalBoundary* and *ThermalOpening*, the *area* property can be replaced by the *area* property of *AbstractThematicSurface*. Likewise, the *surfaceGeometry* describing a GM_MultiSurface can be replaced by the corresponding CityGML 3.0 geometry.

The target classes of the *relatesTo* associations need to be updated with their new CityGML 3.0 counterparts. For *_BoundarySurface* this is *AbstractConstructionSurface*, and for *_Opening* it is *AbstractFillingSurface*. This unveils another problem if the classes were derived from *AbstractConstructionSurface* and *AbstractFillingSurface*. Through the *relatesTo* association, the classes could refer to themselves which is not the purpose. It could be circumvented by adding an OCL constraint or a note prohibiting the self-referral. Either way, it would add more complexity to the Energy ADE specification.

Furthermore, the role name of the association to the former *AbstractConstruction* class is changed to *layering*. Some of the classes in the Material and Construction module need renaming due to semantic discrepancy with the newly introduced construction classes in CityGML 3.0. This is further elaborated in the subsequent subchapter.

4.1.3. Layer and Material module

With CityGML 3.0 the Construction module is newly introduced (see chapter 2.5.1). It contains among others the classes *AbstractConstruction* (superclass of *AbstractBuilding*), *OtherConstruction* and *AbstractConstructiveElement* (see Table 6 for descriptions). On the other hand, a construction according to the Energy ADE solely describes thermal and optical properties of built elements without any geometric context. This can be achieved by specifying these properties for the separate layers made of different materials, which together construct the built element. An example is a double-glazed window, with one layer of glass, a subsequent "empty" layer filled with air, and a second layer of glass.

These considerations reveal the semantic discrepancy between the two concepts of construction. Therefore, a renaming of the Energy ADE construction is necessary. For this, the name *LayeredMaterial* is proposed. It expresses how constructions are composed of several layers made of different materials. Based on this, the classes and role names are changed accordingly. The module itself is renamed to Layer and Material module.

Table 6: Selected CityGML 3.0 construction classes in the Construction and Building module. Taken from Kolbe et al., 2021.

Class	Description
AbstractConstruction	*AbstractConstruction* is the abstract superclass for objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. A connection with the ground also exists when the construction rests by its own weight on the ground or is moveable limited on stationary rails or if the construction is intended to be used mainly stationary.
OtherConstruction	An *OtherConstruction* is a construction that is not covered by any of the other subclasses of AbstractConstruction.
AbstractConstructiveElement	*AbstractConstructiveElement* is the abstract superclass for the representation of volumetric elements of a construction. Examples are walls, beams, slabs.
BuildingConstructiveElement	A *BuildingConstructiveElement* is an element of a Building which is essential from a structural point of view. Examples are walls, slabs, staircases, beams.

As in the other Energy ADE modules, different mapping possibilities are considered and discussed. This is done again with the goal in mind to integrate the ADE as much as possible into the CityGML 3.0 conceptual model.

AbstractLayeredMaterial, LayeredMaterial and ReverseLayeredMaterial

The need for renaming the modules classes already indicates the semantic mismatch with the CityGML 3.0 construction definition. *LayeredMaterial* and its related classes do not constitute a permanent bigger construction element as described in the definitions of Table 6. Therefore, another suitable generalisation class needs to be found.

When integrating the classes into the space and geometry concept, a subclass of *AbstractSpace* has to be chosen. A *LayeredMaterial* complex consist of up to several *Layer* instances, all with a given thickness. Hence, it can be considered a 3D volumetric object. Down the line, none of the more specialised CityGML classes fit semantically. Therefore, the only possible superclass for *AbstractLayeredMaterial* is *AbstractOccupiedSpace*, describing an object blocking space.

However, the general question comes up whether *AbstractLayeredMaterial* should have a spatial extent at all. Although it is an intrinsically physical concept, the classes rather serve the purpose to provide additional relevant information for UBEM. In this context, geometry representations do not add any extra value. For simulations required geometries are already given through the *ThermalBoundary*, *ThermalOpening* and *AbstractBuilding*. Moreover, there are no suitable thematic surfaces that could bound the 3D *LayeredMaterial*.

Based on these arguments, the *AbstractLayeredMaterial* and its specialisation classes remain as in the Energy ADE for CityGML 2.0 on a higher level without any geometric representation. In accordance with the overarching mapping decisions, *AbstractFeatureWithLifespan* is chosen for this.

All the properties and relations, renamed where necessary, stay in place. The *LayeredMaterial* property *serviceLife* is changed from a property by reference to an inline one. This way it is represented in the same way as in *AbstractEnergySystem* or *LayerComponent*. Moreover, *serviceLife* is changed from «type» to «ObjectType» as the former stereotype is no longer applicable for application schemas (see *WeatherData* in 4.1.1 Core module). In this case, «ObjectType» is preferred over «DataType», as it also contains a unique id, enabling referencing through XLinks.

Eventually, the option was discussed to add a relation from *AbstractCityObject* (from *EnergyADECityObjectProperties* in this case) in the ADE Core module to *AbstractLayeredMaterial*. This would enable to further specialise the thermal and optical properties of every city object. Although this might be a valuable addition to the Energy ADE, it exceeds the purpose of a mere mapping by extending the ADE's functionalities. Thus, it is

out of scope for this thesis, but might be considered for future versions of the ADE. The final UML class diagram of the module is depicted in Figure 38.

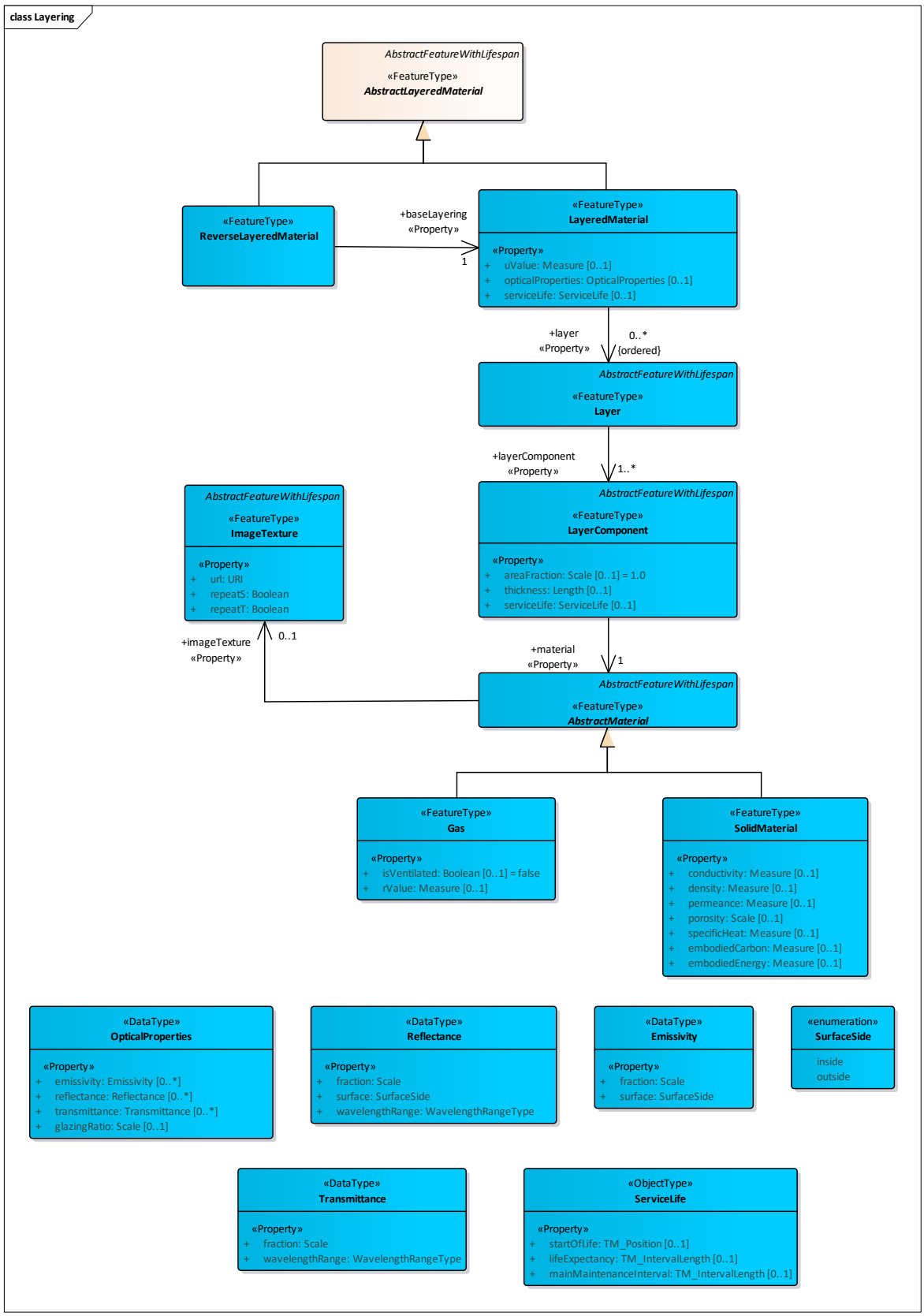


Figure 38: The Energy ADE for CityGML 3.0 Layer and Material module.

Layer and LayerComponent

The same considerations have been applied to the classes *Layer* and *LayerComponent*. For a coherent modelling, they are derived from the same superclass as *AbstractLayeredMaterial* because they ultimately form the *LayeredMaterial*. Therefore, *Layer* and *LayerComponent* are mapped to *AbstractFeatureWithLifespan*.

All the properties and relations remain in place.

AbstractMaterial

AbstractMaterial and its subclasses *Gas* and *SolidMaterial* are sole specifications of the material properties. They are not physically experienceable and can thus not be mapped to *AbstractCityObject* or any of its specialisation classes. Accordingly, it is also derived from *AbstractFeatureWithLifespan*.

4.1.4. Occupant Behaviour module

(Abstract)UsageZone

The *UsageZone* has similar traits as the *ThermalZone* in the Building Physics module. It is a logical building subdivision of volumetric extent which is defined according to a homogenous characteristic, in this case the usage through occupants and facilities. Apparent through this description, it fits the definition of CityGML 3.0's *BuildingUnit* well (see Table 4). However, deriving *AbstractUsageZone* from *BuildingUnit* leads to the same issue of inheriting unwanted properties (see next part on mapping of Energy ADE *BuildingUnit*). Moreover, it would interfere with the aimed logical symmetry between similar classes. Thus, *AbstractUsageZone* is, like *AbstractThermalZone*, subclassed from *AbstractBuildingSubdivision* (see Figure 35 for relation between the CityGML 3.0 classes).

By integrating *UsageZone* in the space and geometry concept, the property *floorArea* can be replaced by the *area* attribute of *AbstractSpace*. Likewise, the *volumeGeometry* is replaced by the solid geometry provided in the CityGML 3.0 Core module.

Furthermore, the *UsageZone* class contains many properties which are described through schedules (*coolingSchedule, heatingSchedule, ventilationSchedule*). The schedules have been

changed from «type» to «FeatureType», which enables the given properties to relate to them through inline relations. A more detailed description on this is given in chapter 4.1.6.

BuildingUnit

The CityGML 3.0 Building module contains a specialisation class of *AbstractBuildingSubdivision* named *BuildingUnit*. According to its definition in the conceptual model standard it is used to describe a logical building subdivision based on a homogenous property such as ownership (Kolbe et al., 2021, see also Table 4). Thus, the description perfectly fits the conception of the Energy ADE *BuildingUnit*. As such, the ADE properties can easily be attached to the CityGML class via the hook mechanism. Due to this reason *BuildingUnit* is mapped as such, even though it hinders *AbstractThermalZone* and *AbstractUsageZone* being subclassed from the same named CityGML class. The alternative is to specialise *AbstractBuildingSubdivision* or CityGML *BuildingUnit* itself to derive the ADE *BuildingUnit* under a changed name. Yet, this could lead to confusion with the *BuildingUnit* class not having the same meaning as before.

Through this mapping, the *BuildingUnit* is furthermore integrated into the space and geometry concept, meaning it can now be modelled with a volumetric extent. In the Energy ADE for CityGML 2.0, *BuildingUnit* is subclassed from *_CityObject*. However, in CityGML 3.0 no concrete class should directly be derived from *AbstractCityObject*. Instead, such a class should ideally be integrated into the new space concept.

Regarding the properties, *floorArea* can be replaced as described before into the *area* property of *AbstractSpace*. The other properties remain in the subclassed ADE DataType *BuildingUnitOccupancy*. Moreover, the relations departing from *BuildingUnit* in the Energy ADE for CityGML 2.0, now derive from *BuildingUnitOccupancy* in order to not alter the CityGML 3.0 data model (see Figure 39).

Occupants and Household

Occupants and *Household* characterise the occupants and their behaviour within a housing unit. As such they do not constitute city objects and are thus mapped to a higher level. Because occupants can change over time, they two classes are derived from

AbstractFeatureWithLifespan. This is also in accordance with the mapping principles stated in the beginning of the chapter.

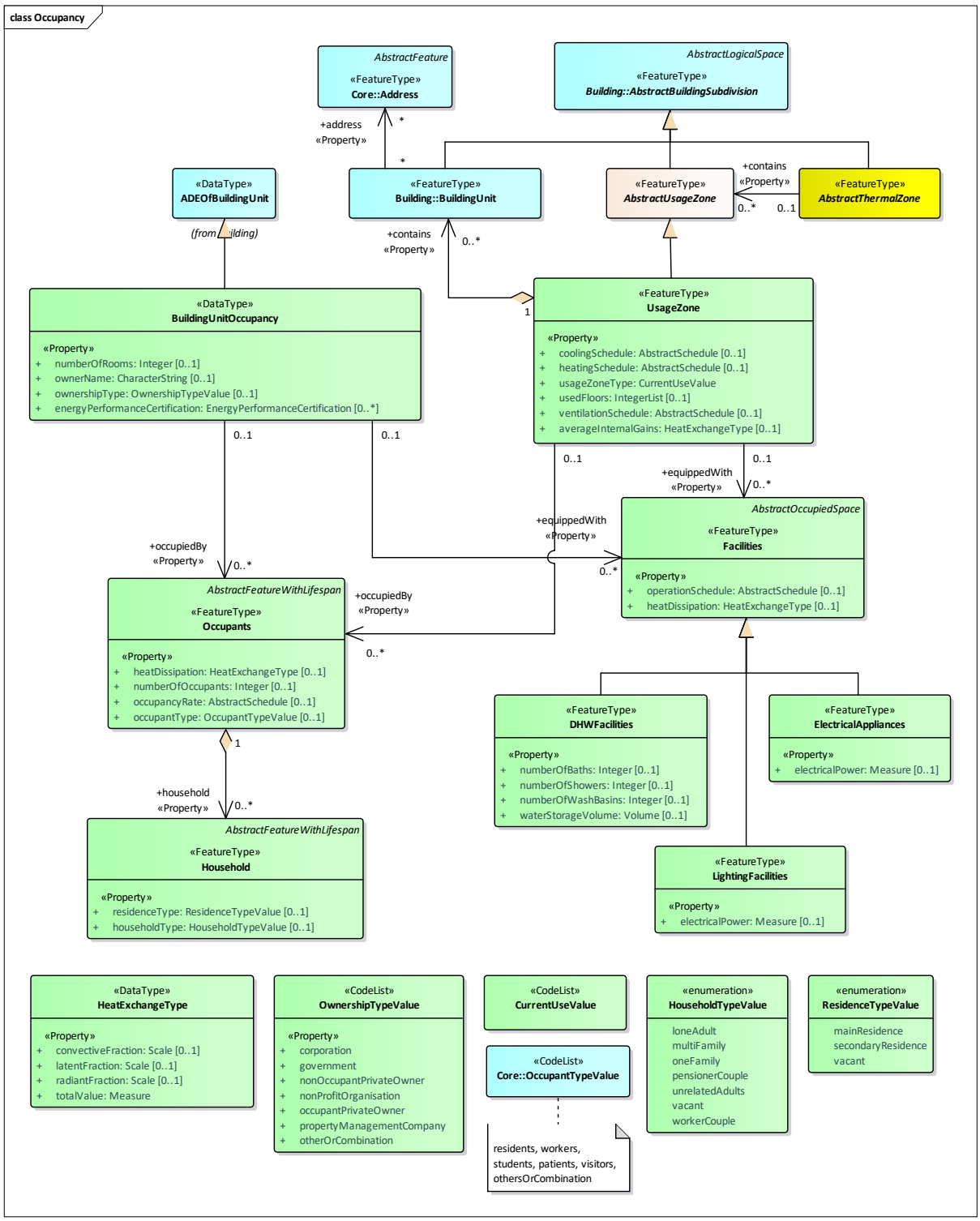


Figure 39: The Energy ADE for CityGML 3.0 Occupant Behaviour module.

All the properties and relations stay in place. Solely the property type of *occupantType* in *Occupants* does not need to be explicitly defined anymore. The codelist with the name *OccupantTypeValue* is now established under the same name and with identical values in CityGML 3.0. Here, it is part of the complex data type *Occupancy*, which is among others used by the *occupancy* property in *AbstractConstruction*. Still, the CityGML 3.0 property can't replace any Energy ADE ones. The ADE classes in question are not specialisation classes of *AbstractConstruction* and thus can not inherit the *occupancy* property.

Facilities

Facilities encompass *DHWFacilities* (Domestic Hot Water, e.g. shower, washbasin), *LightingFacilities* and *ElectricalAppliances*. Like *BuildingUnit*, they derive from *_CityObject* in the Energy ADE for CityGML 2.0. This indicates that *Facilities* are physically experienceable and should thus be integrated into the space and geometry concept when mapping.

Table 7: Selected CityGML 3.0 Furniture and Installation classes in the Construction and Building module. Taken from Kolbe et al., 2021.

Class	Description
AbstractInstallation	*AbstractInstallation* is the abstract superclass for the representation of installation objects of a construction.
BuildingInstallation	A *BuildingInstallation* is a permanent part of a Building (inside and/or outside) which has not the significance of a BuildingPart. Examples are stairs, antennas, balconies or small roofs.
AbstractFurniture	*AbstractFurniture* is the abstract superclass for the representation of furniture objects of a construction.
BuildingFurniture	A *BuildingFurniture* is an equipment for occupant use, usually not fixed to the building.

First, *AbstractInstallation* and *BuildingInstallation* (see Table 7) are considered as superclass for *Facilities*. Bigger built-in facilities such as showers or a bathing tub might fit the definition. But other movable facilities (e.g. a table lamp) do not. Some *Facilities* better match the description of *AbstractFurniture* or *BuildingFurniture* (Table 7). Examples for this are smaller electrical appliances. However, the already mentioned bathtub or lighting facilities which are built into construction surfaces, can hardly be considered as furniture.

These examples show the difficulties of finding a common superclass which fits all kinds of *Facilities*. Therefore, the more generic class *AbstractOccupiedSpace* is chosen for it. Consequently, *Facilities* can be depicted through different geometries as opposed to in the Energy ADE for CityGML 2.0.

4.1.5. Energy Systems module

AbstractEnergySystem and specialisation classes

The mapping of *AbstractEnergySystem* and all its specialisation classes is very similar to the *Facilities* in the Occupant Behaviour module. It is as well derived from *_CityObject* in the Energy ADE for CityGML 2.0. Furthermore, all its subclasses represent some sort of appliances or cables and pipes which are physically present in a building. Therefore, the *AbstractEnergySystem* class is also derived from *AbstractOccupiedSpace*, integrating it in the space and geometry concept. This moreover ensures the logical symmetry to *Facilities*, one of the predetermined mapping principles.

For most of the involved classes nothing changes, except that they can now be geometrically represented. Within the *AbstractSolarEnergySystem*, a subclass of *AbstractEnergyConversionSystem*, the *surfaceGeometry* (multisurface geometry) property can thus be replaced by the corresponding CityGML 3.0 geometry. Beyond this, the *volume* property of *ThermalStorageSystem* (subclass of *AbstractStorageSystem*) needs to be renamed. It refers to the storage volume rather than the object volume which is described through the same called property of *AbstractSpace*. For this reason, the Energy ADE property is renamed to *storageVolume*.

Within the *AbstractEnergySystem* class is also a minor change. The property *yearofManufacture* is replaced by *validFrom* in *AbstractFeatureWithLifespan*. The CityGML property refers to the date when an object was created in the real world. Its corresponding property type *DateTime* requires the specification of the month, day and time. Thus, these values need to be added to the given year.

All the relations in the discussed classes remain as they are. Also the *installedIn* association from *AbstractEnergySystem* to *AbstractCityObject* in the Core module. One of the UML class diagrams is shown in Figure 40, the two remaining ones can be found in Appendix A: UML diagrams of Energy ADE for CityGML 3.0.

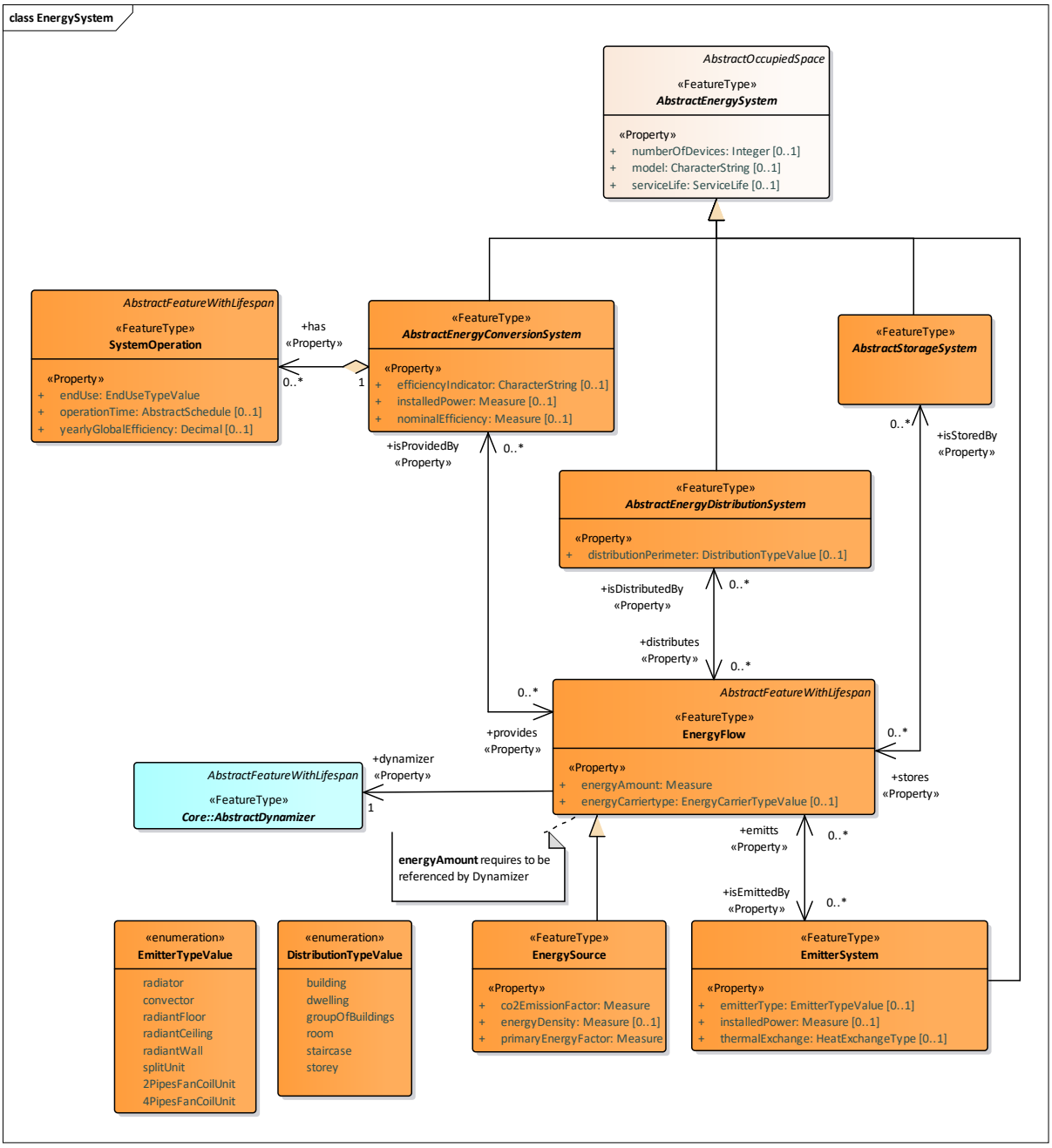


Figure 40: The Energy ADE for CityGML 3.0 Energy System module.

SystemOperation, EnergyFlow and EnergySource

SystemOperation, EnergyFlow and *EnergySource* are the three classes in the Energy Systems module which do not derive from *AbstractEnergySystem*. In the Energy ADE for CityGML 2.0 they are subclasses of *_Feature*. Here, they are mapped to *AbstractFeatureWithLifespan* in accordance with the mapping principles. This ensures that they can be included in different versions of a city model.

Because *EnergyFlow* contains the time-varying property *energyAmount*, a relation to the Dynamizer module need to be established. This is implemented in the same manner as in the Energy ADE Core module.

4.1.6. Time Series Supporting classes

The newly introduced Dynamizer module enables to model time-varying property values and thus shares many similarities with the timeseries supporting classes of the Energy ADE. Due to this overlap, large parts of the timeseries classes are replaced by the Dynamizer module. Although the resulting timeseries module decreases, the same information as before is still conveyed.

How properties are injected with timeseries data through the *Dynamizer* is explained in chapter 2.5.1. At this point it is important to remember that any property in a specialisation class of *AbstractCityObject* can be given time-depending attribute values.

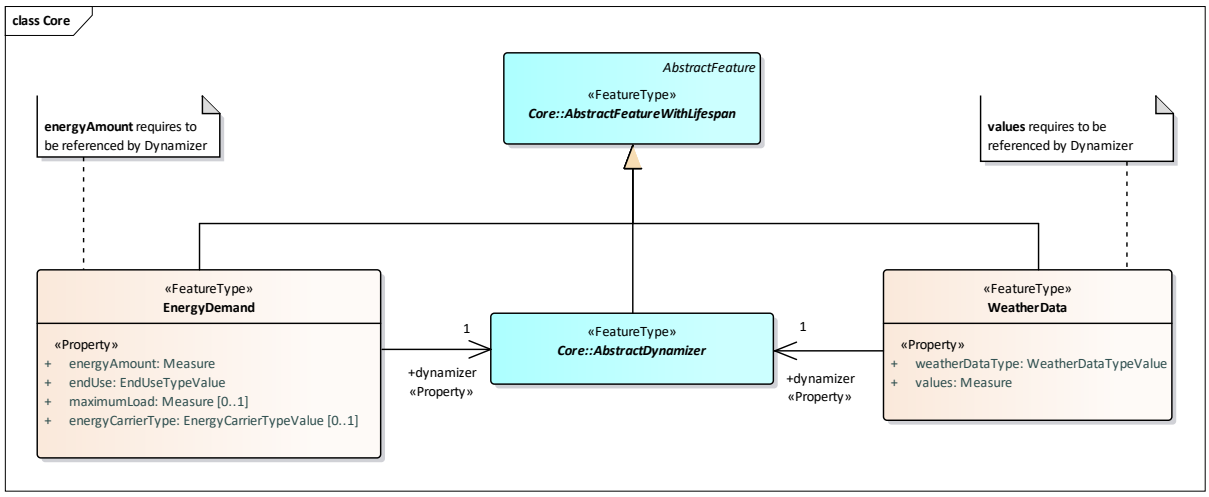


Figure 41: Excerpt of the Energy ADE for CityGML 3.0 Core module, showcasing the UML modelling of time-varying properties.

In the Energy ADE three classes contain such time-varying properties: *WeatherData* (*values*), *EnergyDemand* (*energyAmount*) and *EnergyFlow* (*energyAmount*). All those classes derive from *AbstractFeatureWithLifespan* and have hence no relation to *AbstractDynamizer* through CityGML 3.0. Therefore, this relation is explicitly established by adding an association from the respective classes to *AbstractDynamizer*. Because the properties in question are mandatory, and therefore require to be referenced by the *Dynamizer*, the association multiplicity is set to 1. If the properties were optional, this would not be possible. In fact the multiplicities of the property and the association should be equal. Furthermore, a note is attached to the regarding

Energy ADE classes, informing a user which of the properties requires to be referenced by the *Dynamizer*. Alternatively, OCL constraints could be implemented to enforce the reference. They have the advantage that they can't be ignored like the notes. However, such a constraint would be long and complex, which might be difficult to read and understand for users.

The property type of the time-varying attributes also changes through the new modelling technique. Eventually, timeseries data capture measure attributes (value + uom). Therefore, the property type of the regarding attributes is set to *Measure*. Figure 41 shows how the described modelling is implemented in UML.

TimeValuesProperties

TimeValuesProperties is a complex data type describing metadata of time-dependent properties. In the Energy ADE for CityGML 2.0 it used as property type of the *variableProperties* attribute in *AbstractTimeSeries* (see Figure 17).

The Dynamizer module does not have any equivalent properties to describe this information. Therefore, the *TimeValuesProperties* are added to *AbstractTimeseries*[1] with the ADE hook mechanism. By extending *AbstractTimeseries* instead of *AbstractAtomicTimeseries* the additional properties can also be used for *CompositeTimeseries*.

Within the subclassed data type are directly the *TimeValuesProperties*. This eliminates the intermediary *variableProperties* attribute referring to the *TimeValuesProperties* through its property type.

Moreover, the 0..* multiplicity of *adeOfAbstractTimeseries* prevents the mandatory character of *TimeValuesProperties*. Due to the given multiplicity and the functioning of the hook mechanism, it is not possible for ADEs to include compulsory extension properties. However, as soon as one of the attributes in *TimeValuesProperties* is set, the (other) mandatory properties must also be implemented.

IrregularTimeSeries

The counterpart of *IrregularTimeSeries* is the *GenericTimeseries* in the Dynamizer module. They both describe the same kind of timeseries and are modelled in a similar way. The *uom*

[1] Note the difference between AbstractTimeseries (CityGML 3.0) and AbstractTime**S**eries (Energy ADE).

property in *IrregularTimeSeries* is mapped to the *uom* property of *AbstractAtomicTimeseries*. Conversely, *GenericTimeseries* introduces the mandatory property *valueType* which defines the data type of the measurement values. In the Energy ADE the data type is already predefined as Decimal (see *MeasurementPoint*). Thus, the property value can uniformly be set to double when converting. The time-value pair itself (Energy ADE: *MeasurementPoint* / CityGML 3.0: *TimeValuePair*) is modelled in both cases with a property for the timestamp (*time* / *timestamp*) and one for the value (*value* / *doubleValue*). In CityGML's *TimeValuePair* it is beyond that possible to use further data types, each with their own corresponding property. They are however not relevant in the context mapping or data conversion.

IrregularTimeSeriesFile

Similarly, the *IrregularTimeSeriesFile* is almost completely replaced by the *TabulatedFileTimeseries*. The *uom* property is again mapped to the *uom* property of *AbstractAtomicTimeseries*. Except for *recordSeparator*, all properties are assimilated by their counterparts in *TabulatedFileTimeseries*. *recordSeparator* is incorporated to the CityGML 3.0 class through the hook mechanism. Additionally, *TabulatedFileTimeseries* contains the compulsory property *fileType* specifying the external file's format. This value needs to be set accordingly when converting.

CityGML 3.0 provides next to the *TabulatedFileTimeseries* another class for externally stored timeseries. *StandardFileTimeseries* handles files encoded in special standardised formats. Examples are the OGC TimeseriesML or OGC Observations & Measurements Standard (Kolbe et al., 2021). Nonetheless, this class is not considered for mapping as many properties would be lost or the given input file would need to be altered in order to match the standards.

RegularTimeSeries

For the *RegularTimeSeries* exists no ideal equivalent in the Dynamizer module. One possibility is to nonetheless map it to *GenericTimeseries* and derive the timestamps through the given temporal extent and the interval length. But this eradicates the benefit of a more compact encoding through the *RegularTimeSeries* class.

Therefore, a new class is created for it, deriving from *AbstractAtomicTimeseries*. The name is adapted to *RegularTimeseries* to match the other Dynamizer classes. Moreover, the property

temporalExtent is replaced by the properties *firstTimestamp* and *lastTimestamp* of the class *AbstractTimeseries*.

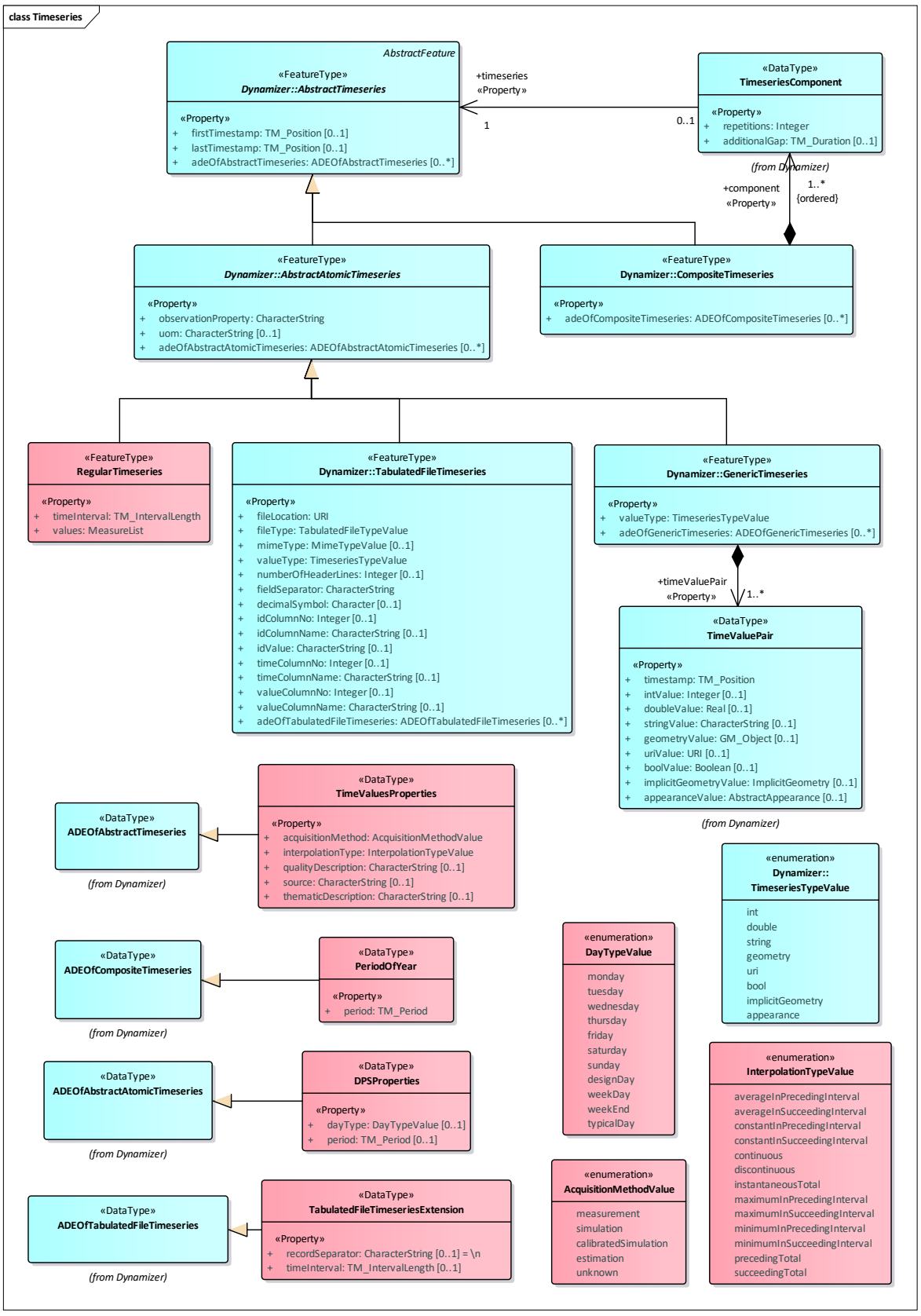


Figure 42: The Energy ADE for CityGML 3.0 Time Series module.

RegularTimeSeriesFile

Mapping the *RegularTimeSeriesFile* faces the same problem as the *RegularTimeseries*, it has no counterpart within the Dynamizer module. The *TabulatedFileTimeseries* enforces with OCL constraints, that either *timeColumnNo* or *timeColumnName* require a value. In concrete terms this means that a column containing the timestamps needs to be specified. Though, such a column is not part of a regular timeseries file.

In the following, five options on how to integrate the *RegularTimeSeriesFile* in the Dynamizer module are presented.

Option 1: Change input file

The first option is to alter the input file in a way that it contains timestamp values. The information can be derived through the temporal extent and the time intervals. This has the advantage that no additional class is created. On the other hand, input files need to be manipulated priorly, which is potentially a time-consuming process. Consequently, it eliminates the possibility to connect any true regular timeseries files to the Energy ADE. The resulting loss of functionalities contradicts the research goal of a lossless mapping which is why the remaining options are favoured.

Option 2: Create own ADE class

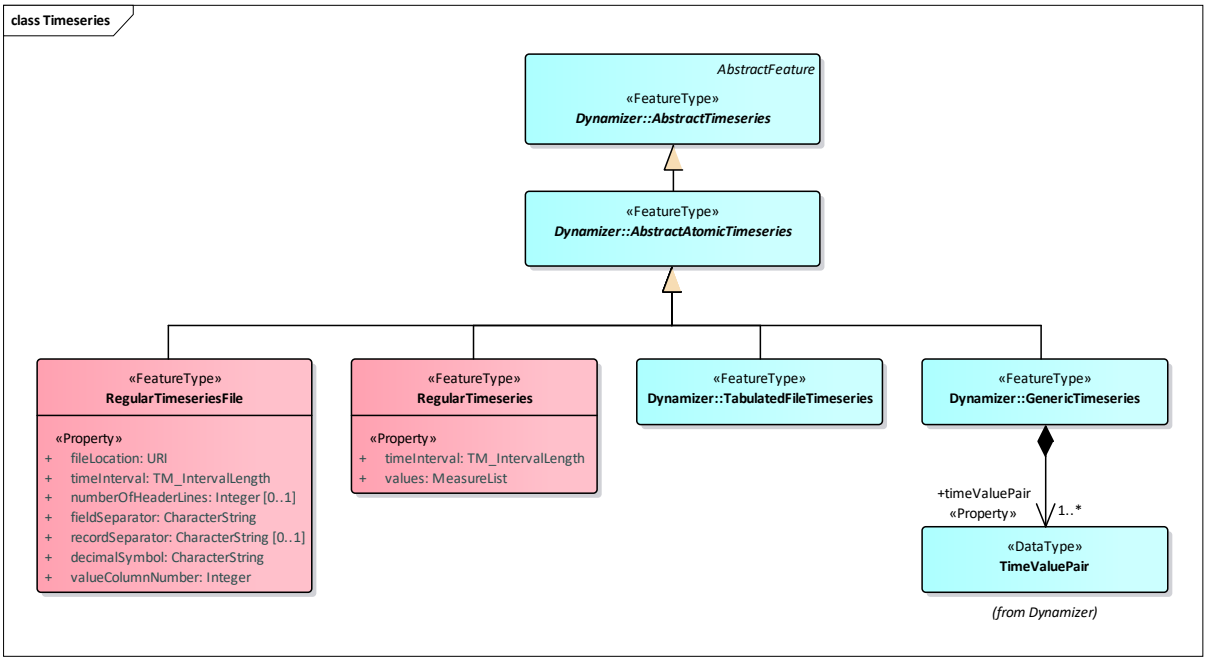


Figure 43: Option 2 – Creating a new ADE class to map RegularTimeSeriesFile to CityGML 3.0.

Alternatively, the second option is to create an own ADE class for *RegularTimeseriesFile*. The characteristics are similar to *RegularTimeseries* and *TabulatedFileTimeseries*. It derives from *AbstractAtomicTimeseries*, the properties *uom* and *temporalExtent* are replaced by CityGML 3.0 ones. How this option looks like in terms of UML is depicted in Figure 43. This solution is straightforward and easy to use. Yet, it models repetitive information which are in sum already contained in the other specialisation classes of *AbstractAtomicTimeseries*.

Option 3: One common ADE class for *RegularTimeSeries* and *RegularTimeSeriesFile*

Another possibility is to create a common class for *RegularTimeSeries* and *RegularTimeSeriesFile.* It contains the union of the original classes' properties. The common properties have a multiplicity of 1, whereas the remaining ones are set to 0..1. Additionally, a new mandatory property (*isFile*) of type Boolean is introduced to specify how the information is stored. Based on the property value, OCL constraints determine which properties need to be at least set (see Figure 44).

This option overcomes the downside of the previous one by avoiding repetitive modelling through two separate classes. Although it comes at the cost of being more difficult to understand by a user, especially because of the OCL constraint. Moreover, the class name hides the fact it also stores timeseries files. In such a property-based approach, the stored object only unveils through the properties. Opposed to that, CityGML 3.0 relies on an object-based approach in which the class name plays a vital part in object definition.

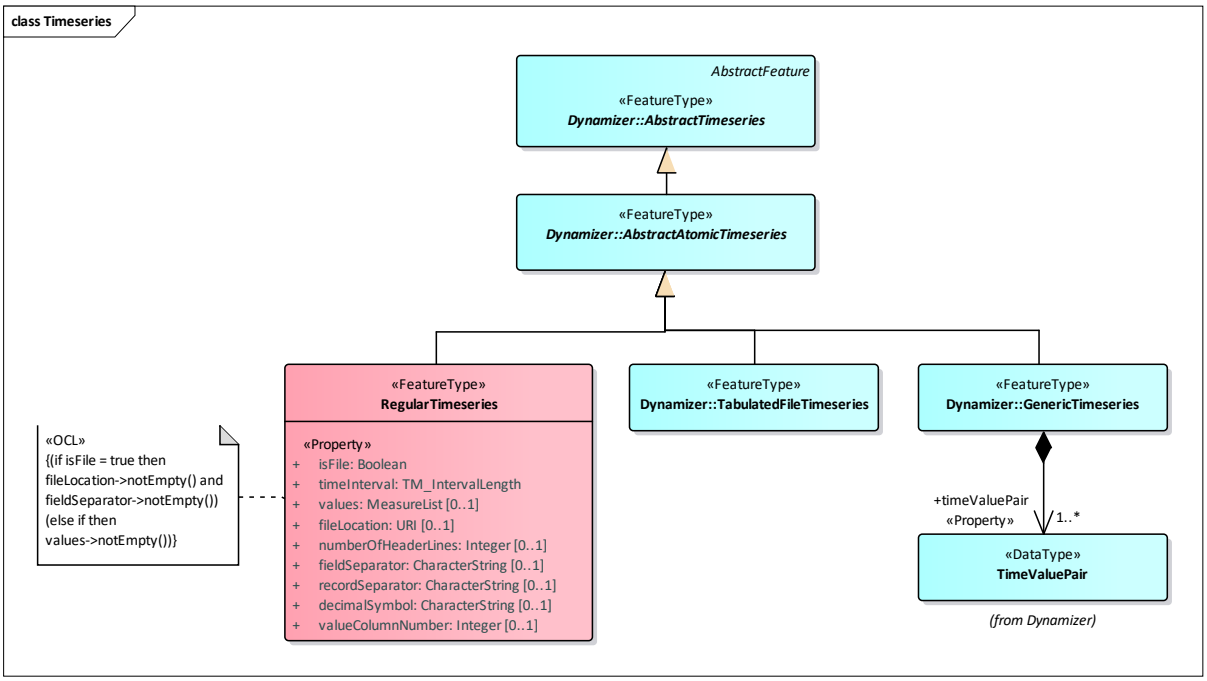


Figure 44: Option 3 – Creating one common ADE class for RegularTimeSeries and RegularTimeSeriesFile.

Option 4: Introduce an *AbstractRegularTimeseries*

The fourth option follows the object-based approach by creating an abstract class for *RegularTimeseries* and *RegularTimeseriesFile*, containing their common property *timeInterval* (see Figure 45). This emphasizes their commonality while acknowledging their different purposes.

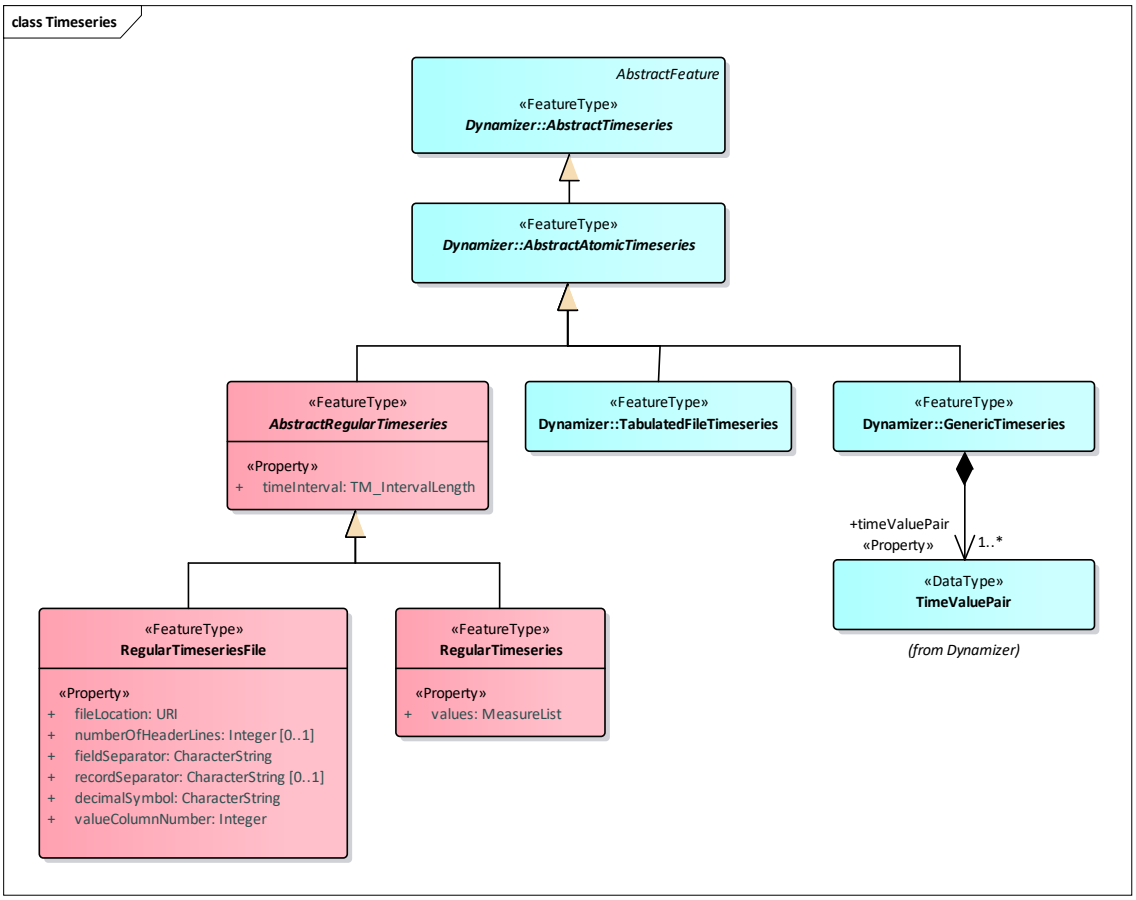


Figure 45: Option 4 – AbstractRegularTimeseries with specialisation classes for RegularTimeseries and RegularTimeseriesFile.

Option 5: Integrate *RegularTimeSeriesFile* in *TabulatedFileTimeseries* with workaround

In this last option a workaround for the OCL constraint of *TabulatedFileTimeseries* is explored. Although a timestamps column is non-existent in regular timeseries files, the property *timeColumnName* asking for a CharacterString value can be given a NaN value. This value can express in form of a string that such a column is not included. The property *recordSeparator* is already included through the ADE hook. To this, the property *timeInterval* is added with a 0..1 multiplicity. Furthermore, as already shown in the other classes, the properties *uom* and

temporalExtent are replaced by CityGML 3.0. The implementation of this in terms of UML is depicted in Figure 42.

Option 5 has the advantage of reusing existing classes instead of creating new ones. This keeps the UML model compact and avoids modelling redundant information. It furthermore adheres to the mapping principles of integrating the Energy ADE as much as possible into CityGML 3.0. Based on these arguments, this last option is finally chosen for the *RegularTimeSeriesFile*.

4.1.7. Schedules Supporting classes

In the Energy ADE for CityGML 2.0 timeseries and schedules function in the same way. Because the timeseries are largely replaced through classes in the Dynamizer module, its operating principles are adapted. This comes at the cost of a more complex implementation as compared to the Energy ADE already in place. However, the schedules are not bound to any pre-existing classes in CityGML 3.0 because there is nothing alike in place. Therefore, they can independently be mapped with a simpler method than the timeseries.

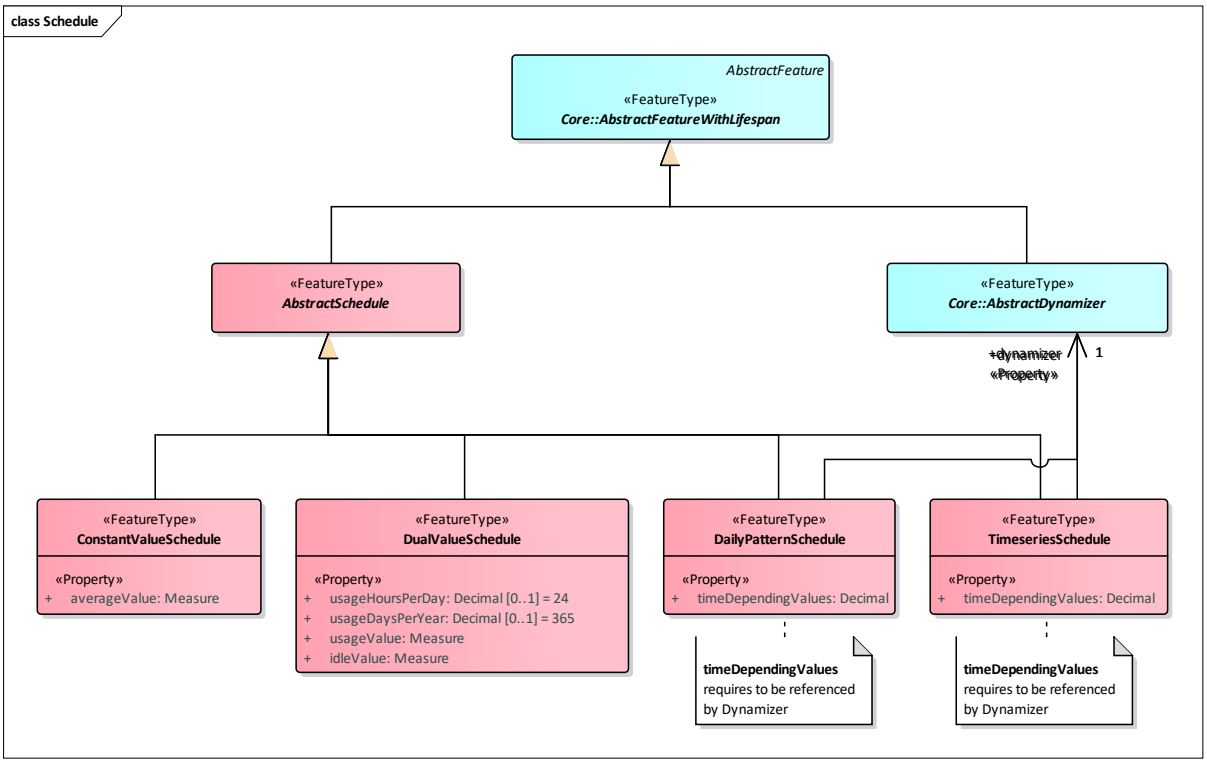


Figure 46: The Energy ADE for CityGML 3.0 Schedules module.

As described for *WeatherData* in chapter 4.1.1. the stereotype «type» is not utilised for application schemas anymore. Consequently, the stereotype of *AbstractSchedule* and its

subclasses needs to be changed. «DataType» is ruled out due to its missing unique id. The id is necessary for referencing schedules with XLinks, a in this context frequently used method. Thus, *AbstractSchedule* is converted to «FeatureType» preserving the unique id.

The new features are integrated into CityGML 3.0 by deriving them from *AbstractFeatureWithLifespan*. *AbstractCityObject* as alternative is incompatible because schedules are not considered a city object nor have a spatial extent. *AbstractFeature* on the other hand is a possible option, yet *AbstractFeatureWithLifespan* is preferred due to the general modelling principles. Coherent with the remaining ADE classes, this enables to model different versions of the schedules. Furthermore it fulfils the guideline of logical symmetry when mapping, as both timeseries and schedules eventually derive from the same class. Figure 46 shows the UML diagram of the schedules and how the classes are related to the Dynamizer module.

The properties being described through schedules, have the property type *AbstractSchedule* (see Figure 47 left). This inline representation is equal to an explicit relation (by reference) from the respective feature type to *AbstractSchedule* (see Figure 47 right).

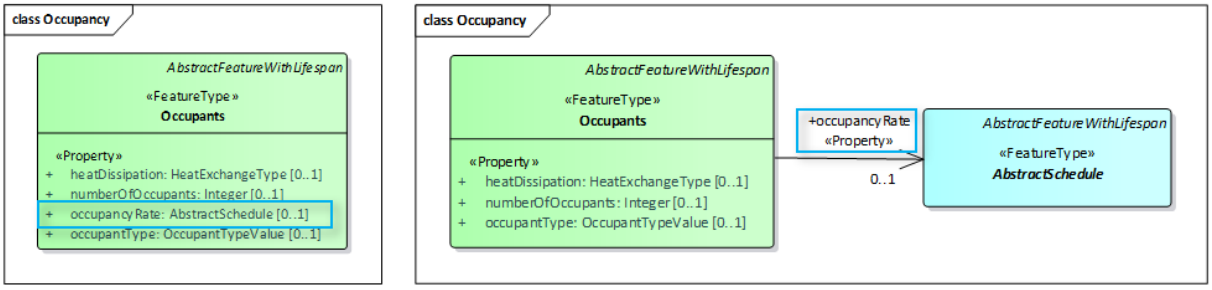


Figure 47: occupancyRate with implicit relation (left) and explicit relation (right) to AbstractSchedule.

ConstantValueSchedule and DualValueSchedule

Both classes are converted to feature types and derive from *AbstractSchedule*. Besides that, they keep their properties as before.

TimeseriesSchedule

Similarly, the *TimeSeriesSchedule* remains relatively unchanged. Its name is adapted to *TimeseriesSchedule* according to the CityGML 3.0 practice. Additionally, the property type of *timeDependingValues* is changed to Decimal. Opposed to the other properties carrying time-

varying values with measure types, the *TimeseriesSchedule* does not require a unit of measure. The values rather describe a ratio how intensely something is used at a given time. In order to write those time varying values, a connection from *TimeseriesSchedule* to *AbstractDynamizer* is added. As in examples before, the role name is set to *dynamizer*, and the multiplicity to 1.

DailyPatternSchedule

To map the *DailyPatternSchedule* there are two possibilities. The first one being to keep the schedule as it is, modelling it through compositions *PeriodOfYear* and *DailySchedule*. The second option is to make use of *CompositeTimeseries* and *TimeseriesComponent* in the Dynamizer module. A *CompositeTimeseries* can have several *TimeseriesComponents*, whereas each of them is described through one of the given timeseries (Figure 48). As such, the provided structure allows to construct more complex, nested timeseries, similar to the *DailyPatternSchedule*.

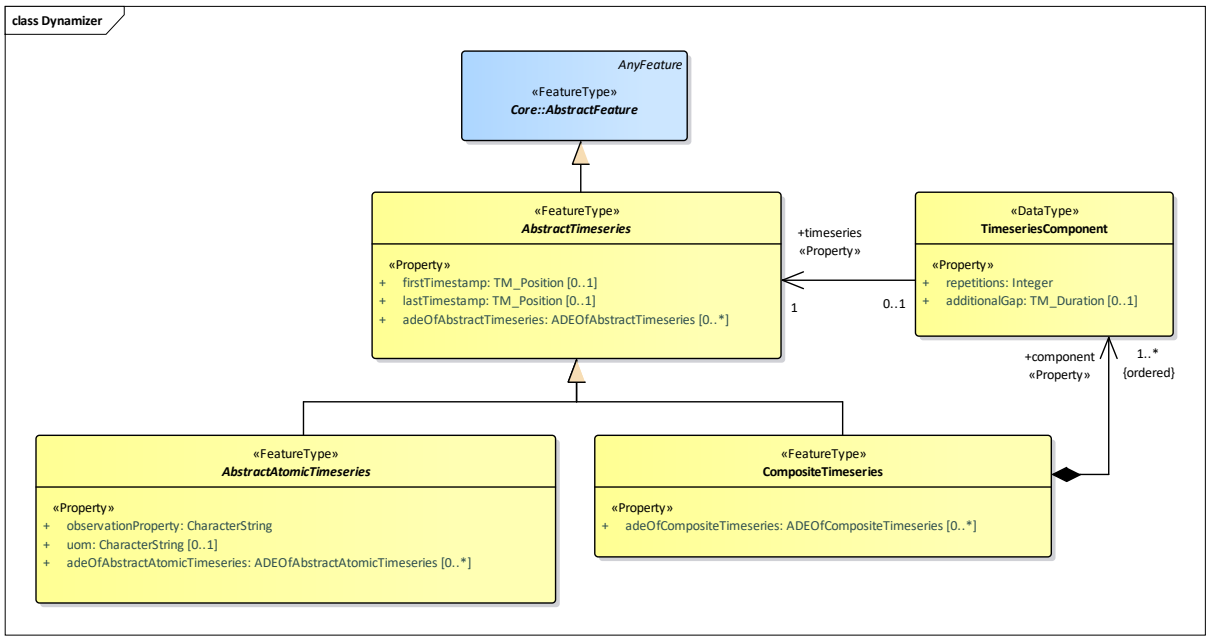


Figure 48: CompositeTimeseries and TimeseriesComponent in the Dynamizer module of CityGML 3.0. Adapted from Kolbe et al., 2021.

For a lossless mapping, some information from the Energy ADE's *DailyPatternSchedule* and its compositions needs to be added to the Dynamizer classes. The «dataType» *DailySchedule*, carrying the actual time-varying property (*schedule*) in the Energy ADE for CityGML 2.0, can be corresponded to the «DataType» *TimeseriesComponent*. However, the second property (*dayType*) cannot be added to it, as classes with the stereotype «DataType» can't be extended with the usual CityGML extension mechanisms. However, because *TimeseriesComponent* is

eventually described through other timeseries, *dayType* can be added to *AbstractAtomicTimeseries* via the hook mechanism.

Next is the *period* property in the Energy ADE for CityGML 2.0's *PeriodOfYear*. In case the timeseries are all within one time period, the *period* property can be added with the hook to the CompositeTimeseries. The class entails all the other timeseries and thus the *period* then accounts for all of them. Alternatively, if there are several *PeriodOfYear* within one *DailyPatternSchedule*, the period can also be written to the corresponding timeseries directly. To do this, the period is additionally added with the hook to *AbstractAtomicTimeseries*. This solution offers a shortcut, as otherwise such cases would need to be modelled through *CompositeTimeseries* consisting of *CompositeTimeseries*. Figure 49 summarises the with the ADE hook incorporated properties in order to map the *DailyPatternSchedule*.

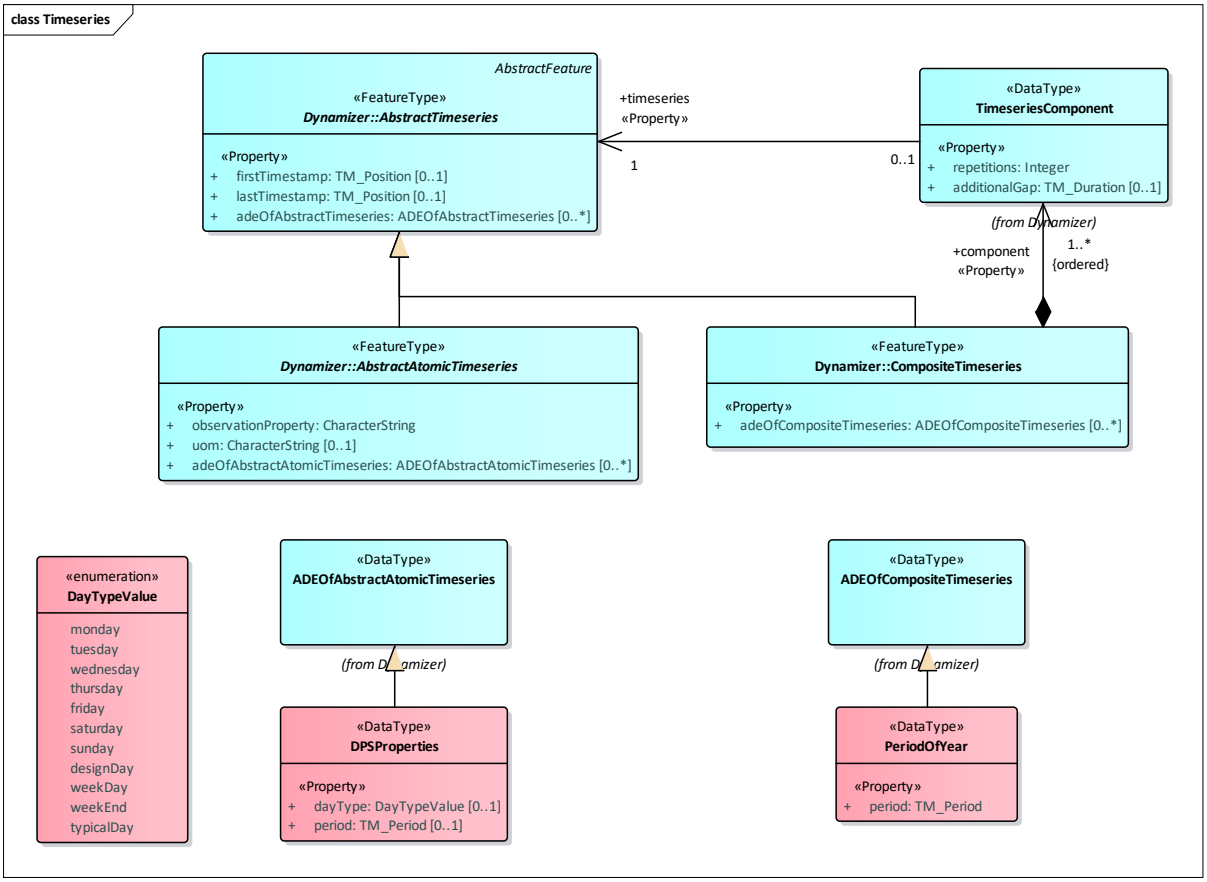


Figure 49: Excerpt of the Energy ADE for CityGML 3.0 timeseries module, showing the extensions relevant for the DailyPatternSchedule.

Through the described mapping, only one property (*timeDependingValues*) containing the time-depending values is needed in the *DailyPatternSchedule*. Additionally, a relation to *AbstractDynamizer* is needed in the same way as for the *TimeseriesSchedule* (Figure 46). This

mapping option adheres to the mapping principle to integrate the Energy ADE as much as possible into CityGML 3.0. Thus, it is preferred and implemented one.

4.1.8. Weather Data supporting classes

The Weather Data module contains besides *WeatherData,* which is discussed under the Core module in chapter 4.1.1, the class *WeatherStation*. In the Energy ADE for CityGML 2.0 it derives from *_CityObject* and is thus integrated into the space and geometry concept when mapping. It is not defined what form a *WeatherStation* exactly takes. It can be in form of its own building or also just an assembly of different measurement instruments. Therefore, the class is kept higher up in the CityGML 3.0 hierarchy at *AbstractOccupiedSpace*.

Due to its integration in the space concept, the point geometry *position* is no longer needed. The property *stationName* on the other hand remains as it is, although it is not strictly necessary as the name can also be conveyed through the *name* property of *AbstractFeature*. Moreover, the *parameter* association to *WeatherData* is omitted due to a mistake in the Energy ADE. Because *WeatherStation* is a subclass of *AbstractCityObject*, and any *AbstractCityObject* already has a connection to *WeatherData* (*weatherData*), the association is redundant. The module is depicted in Figure 50.

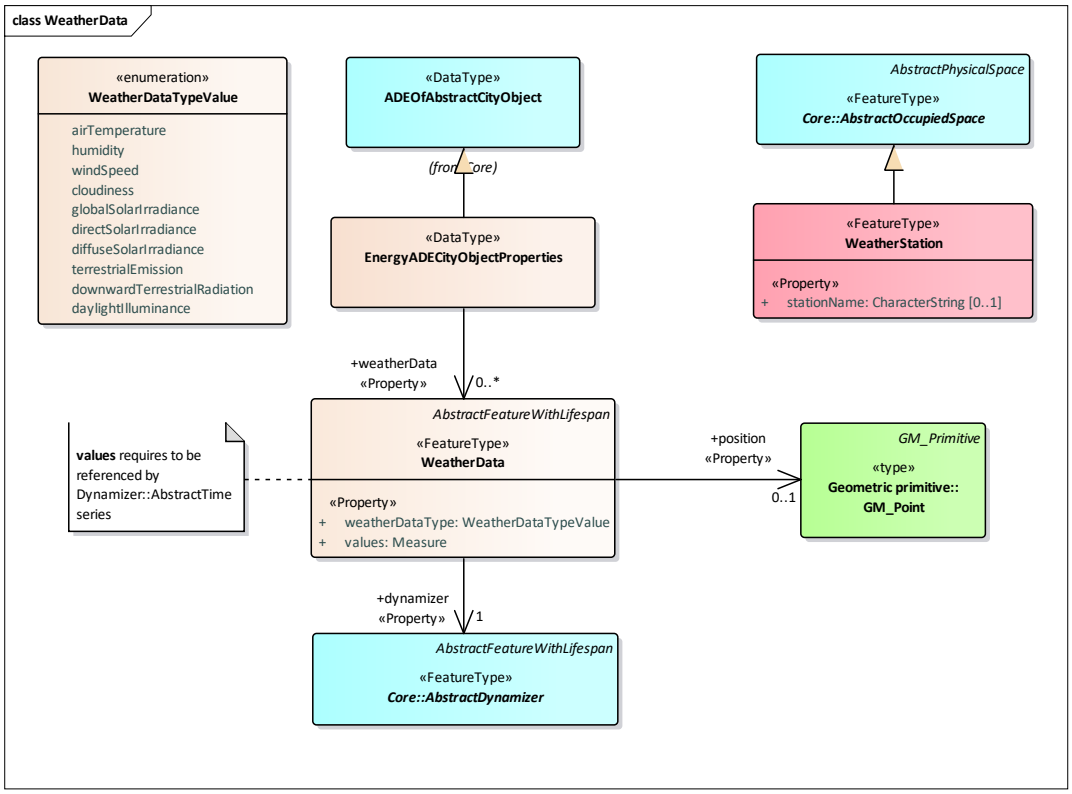


Figure 50: The Energy ADE for CityGML 3.0 Weather Data supporting classes.

4.2. Derivation of the XSD schema file

In order to produce valid Energy ADE data, an according schema file is needed in the background. This schema is automatically derived from the UML diagrams through ShapeChange.

At the heart of the tool is a to the use case customised configuration file. For this thesis, the configuration file of the UtilityNetwork ADE for CityGML 3.0 (citygml3-utility-network-ade, 26.11.2022) is used as a starting point. Both ADEs extend CityGML 3.0 in a GML target encoding, and thus make use of the same namespaces and encoding rules. Therefore, only small parts of the configuration file need to be adapted in order to fit the Energy ADE.

Generally, the configuration file specifies which UML diagrams are processed, how they are supposed to be encoded and where to store the results. First, some input parameters are defined. Among them are the path to the EA file (`parameter name="inputFile"`) and the regarding application schema (`parameter name="appSchemaName"`). Furthermore, the required stereotypes are laid out, including standard GML ones and the ones specific to CityGML 3.0 (e.g. TopLevelFeatureType). Next, the output directory for the log file (`parameter name="logFile"`) and the resulting schema (`targetParameter name="outputDirectory"`) is determined.

An especially important part of the configuration file is the targets definition. It includes the target encoding, which is an XML schema file (XSD) in this case, and the applied encoding rules. The encoding rules are based on GML 3.2.1 and are extended by user defined ones (`EncodingRule name="citygml" extends="iso19136_2007"`). Moreover, the CityGML 3.0 namespaces are given in order to link them correctly in the ADE. Additionally, it is defined how the applied data types in the UML diagrams are mapped to xml ones. All the standard data types are imported through a link provided by ShapeChange (`xi:include href="http://shapechange.net/resources/config/StandardMapEntries.xml"`). However, the mapping rules for user defined data types need to be individually specified (`xsdMapEntries`). For the Energy ADE this is only the case for the type IntegerList used by property *usedFloors* in *UsageZone*. Finally, it is specified whether codelists are encoded, and if so, in which output directory they are saved.

After the configuration file is set, ShapeChange itself is executed in a command line interface. The command includes the used java version, the path to the ShapeChange installation and the configuration file.

With the execution of ShapeChange comes a validation of the UML diagrams against the given ISO standards. Any errors or warnings are summarised with a short description in the log file. As a consequence, ShapeChange also constitutes an important control instance while creating the UML diagrams.

Eventually, the final XSD schema file is manually reviewed by checking whether all classes and properties are correctly modelled. The XSD schema of the UtilityNetwork ADE served in this step as a guide for a better understanding of the file's structure.

4.3. Conversion to Energy ADE for CityGML 3.0

The conversion to Energy ADE for CityGML 3.0 is realised with a workspace in FME. In order to create and test it, a suitable input dataset is required. This is generated by the FME workspace described in the first part of the chapter. The conversion workspace itself is subject to the latter part.

4.3.1. Test data creation

As starting point for the test data creation serves an already with Energy ADE enriched CityGML 2.0 dataset. It is an imaginary city model consisting of 12 buildings modelled through their boundary surfaces *RoofSurface*, *WallSurface* and *GroundSurface* in LOD 2. Additionally, the buildings are represented through a *lod0FootPrint* and a *referencePoint* respectively. Each building contains one *UsageZone* and one *ThermalZone* with their boundaries following the respective CityGML boundary surfaces. Thus, the *ThermalBoundary* always follows the building geometries whereas the *ThermalOpenings* are modelled without geometry. Additionally, they are described through *Constructions* with *Layers*, *LayerComponents* and *Materials*. Furthermore, every building has a set of *Facilities*, *Occupants* and *Households* as well as a timeseries for *EnergyDemand* and an *occupancyRate* schedule. Lastly, the city model also contains a *WeatherStation* for temperature and humidity on top of Building 1.

This shows that many ADE feature types and properties are already present in the dataset. However, the entire Energy Systems module is missing and thus needs to be added. Moreover, some individual feature types are not included yet (*BuildingUnit, ReverseConstruction, ImageTexture, Gas*) as well as a few isolated properties (e.g. *opticalProperties* and *serviceLife* in *Construction*, *openableRatio* in *ThermalOpening*). Additionally, only *RegularTimeSeries* and *DailyPatternSchedule* are applied out of the Time Series and Schedules supporting classes. All of the missing features and properties are complemented through the FME workspace.

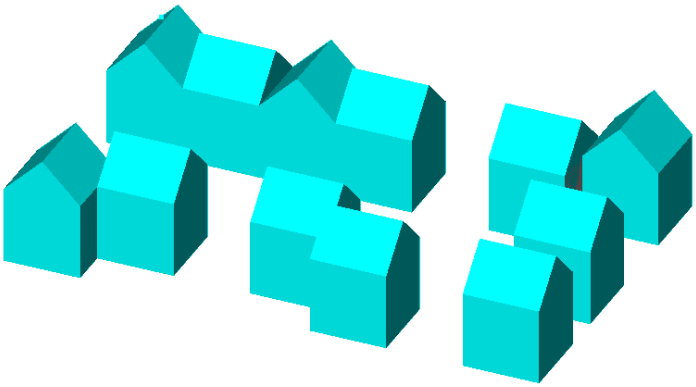


Figure 51: The test dataset in boundary representation. Visualisation by FZK Viewer.

In a new workspace, a CityGML 2.0 Reader and Writer are inserted with the additional Energy ADE schema file. This imports a separate Reader and Writer Instance for every feature type of CityGML 2.0 and the Energy ADE. Within the settings, the option “validate CityGML Dataset File” is set to true. In the Reader properties, the number of maximum nested attributes is set to 3000 in order to also expose the FME encoding for long and nested properties. This is especially useful when modelling the remaining timeseries and schedules.

The enrichment of the dataset is implemented through so called transformers in FME. Generally, three scenarios of how the input is processed can be differentiated. They are schematically displayed in Figure 52 within the blue boxes.

In Scenario A, the feature types which are already complete in terms of their properties and relations are simply connected to their respective writers. Scenario B describes the case if the feature is already present in the input, but some properties are missing, or additional possibilities (e.g. for the schedules) are included. Here, the attributes are manipulated with an AttributeManager and/or an AttributeCreator transformer before connecting it to the Writer. The AttributeCreator only comes into place for creating list attributes. In FME, list attributes are used for properties or relations with a 0..* multiplicity.

The last scenario applies if a feature type is not present yet within the input data. Through a Creator transformer, an instance of the respective feature is created. By joining the Reader and the creation instance at the subsequent AttributeManager, all possible properties already show up and only need to be assigned property values.

In some cases additional filter transformers are applied to separate only one specific instance and manipulate its attributes individually. This is for example the case for the *EnergyDemand*, where one instance is separated to model a *IrregularTimeSeries*.

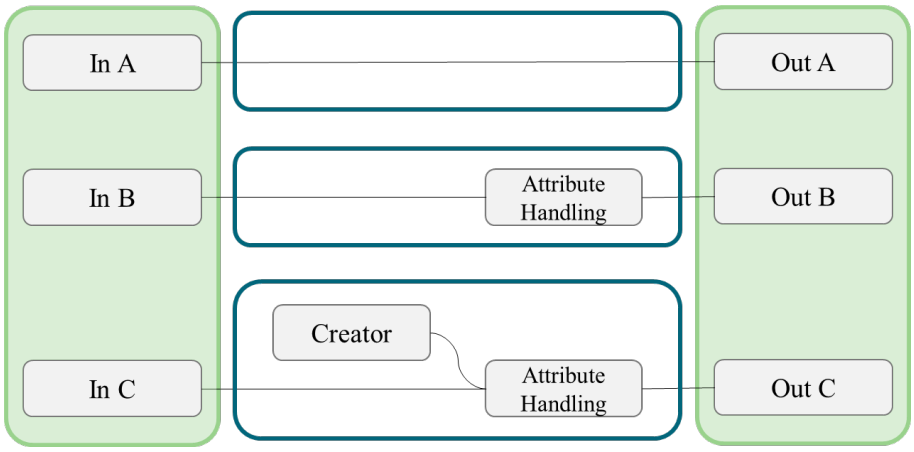


Figure 52: Schematic representation of the FME workspace to create test data. Left in green are the Reader features, right the Writer features. A, B and C refer to the three scenarios how the data is processed.

4.3.2. Conversion workspace

The conversion to Energy ADE for CityGML 3.0 builds up on an FME workspace for converting the Building module to CityGML 3.0 (see Method). As in the workspace before, the data is imported with a CityGML Reader. However, it only reads and writes up to CityGML 2.0. Therefore, to export the converted data, a generic GML Writer is in place. By providing the CityGML 3.0 XSD schema files, the new standard can already be exported. The same accounts for reading CityGML 3.0 with the GML Reader.

Because the Reader and Writer in the workspace are only set up for CityGML features, they have to be updated with the "Update Feature Types" option in FME to also incorporate the Energy ADE. This imports the necessary ADE feature types for further processing.

The conversion workspace is schematically depicted in Figure 53. All over, the conversion consists in large parts of renaming the attributes due to their changed FME encoding. Furthermore, the ADE geometries have to be adapted to the CityGML 3.0 ones (Figure 53, In/Out A). Also new is that the timeseries, the schedules and *WeatherData* now have their own Writer feature types. Therefore, the regarding information needs to be extracted and processed accordingly (Figure 53, In B/Out Schedule FT, Dynamizer FT). Finally, the conversion handles individual changes of some mapped properties and property values. Everything concerning general CityGML 3.0 and thus, elements already implemented beforehand in the workspace, are not discussed separately.

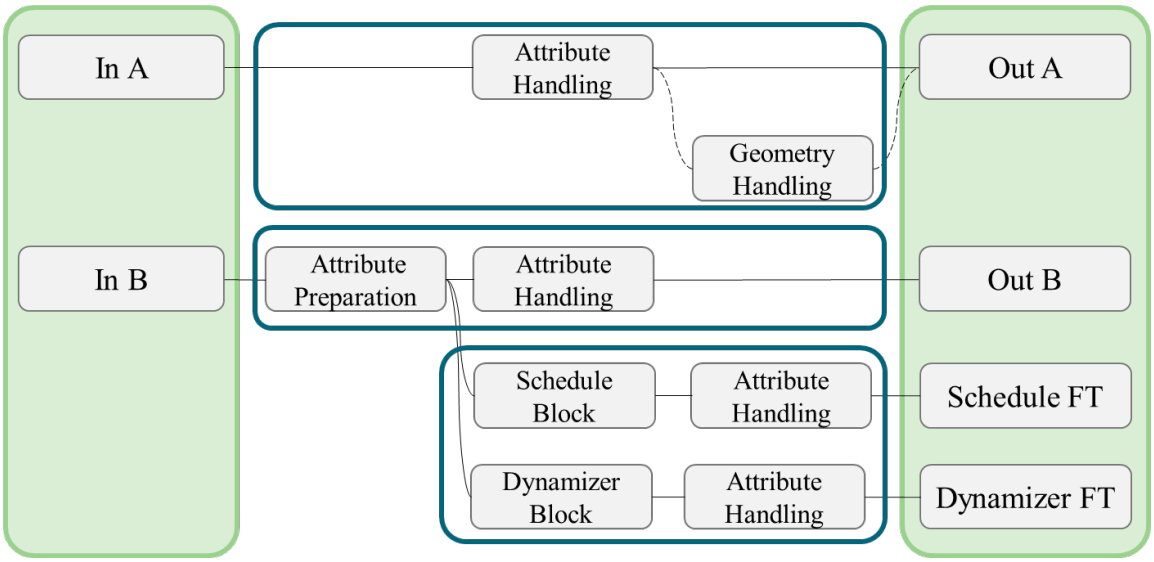


Figure 53: Schematic representation of the FME workspace to convert the data to CityGML 3.0 + Energy ADE. Left in green are the CityGML Reader features, right the GML Writer features In B contains timeseries and schedule information which is extracted and then written to its according Writer feature.

Attribute Renaming

There are a few things that the processing of all feature types have in common. Among them are the attribute names, how they are displayed and encoded in FME, differ between the CityGML Reader and the GML Writer. The example of two *Occupants* properties in the table below demonstrates this. In the CityGML Reader the name fractions are separated by underscores. Each property, also in case of nested properties (see second example in Table 8), is preceded by "energy_". Moreover, everything is written in lowercases. The GML Writer encoding on the other hand, resembles the given property names in the UML diagrams, also in terms of capitalisation. Nested properties are simply separated by full stops (.).

Table 8: Different property encodings in FME through CityGML Reader / Writer and GML Reader / Writer. At the examples of numberOfOccupants and heatDissipation of the Occupants feature type.

CityGML Reader / Writer	**GML Reader / Writer**
energy_number_of_occupants	numberOfOccupants
energy_heat_dissipation_energy_heat_exchange_type_energy_convective_fraction_units	heatDissipation.HeatExchangeType.convectiveFraction.uom

Through the capitalisation of the nested attributes, and because not every "." in the updated encoding translates to an "_energy" in the original one, it is difficult to develop a general rule fitting to all properties. Therefore, the attributes are individually renamed for every feature type

with an AttributeRenamer transformer in FME. Note that gml attributes are automatically interpreted correctly by the GML Writer, which is why properties such as *gml_id* or *gml_parent_id* do not require renaming.

List attributes in general have to be renamed with a ListRenamer transformer. The same encoding rules apply as for regular attributes. Renaming *gml_name* through a regular AttributeRenamer is an exception, as the list entry can only be 0 and can thus be hardcoded.

Another small detail is that the *citygml_feature_role* in *UsageZone*, *ThermalZone* and the schedule properties of *UsageZone* needs to be renamed to *gml_parent_property* in order to be written correctly. In the first two cases, the classes would instead be written with the role *buildingSubdivision* due to the relation between *AbstractBuilding* and *AbstractBuildingSubdivision*. In the case of the three schedule properties in *UsageZone* (*coolingSchedule, heatingSchedule, ventilationSchedule*), all the schedules would be written to coolingSchedule and thus violating the given multiplicity of 0..1.

Geometries

The Energy ADE geometries are, with the exception of the *WeatherData position*, fully replaced by CityGML 3.0 geometries. What eventually changes is only the role name pointing to the geometry. This is remodelled in FME by altering the geometries name and traits.

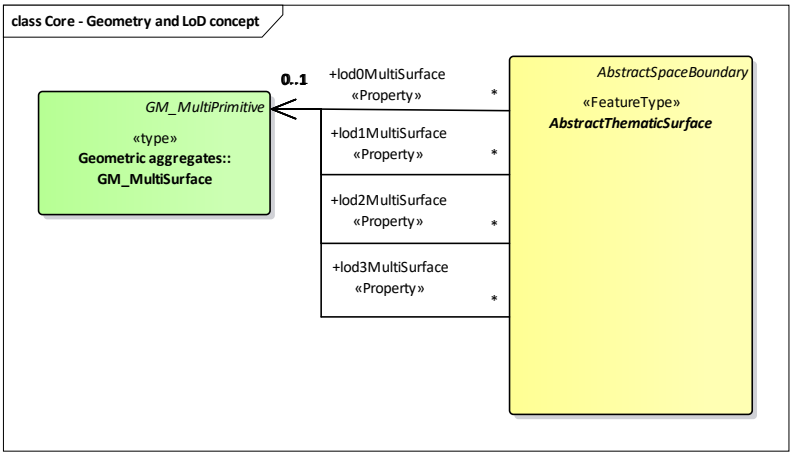


Figure 54: MultiSurface representation of AbstractThematicSurface in the CityGML 3.0 Core module.

ThermalBoundary and its MultiSurface geometry serve therefor as an example. However, the same methodology with adapted parameters applies to all other geometries.

As it can be taken from the Energy ADE for CityGML 2.0 UML diagram (Figure 13), the geometry of *ThermalBoundary* is referenced via the *surfaceGeometry* property. In CityGML

3.0, the corresponding role name is *lodxMultiSurface*. It connects *AbstractThematicSurface*, of which *ThermalBoundary* is a subclass, to the MultiSurface geometry (see Figure 54).

In FME this updated role name has to be set as geometry name and geometry trait at the MultiSurface level. First, a new attribute *citygml_lod_name* is created with the value *lod2MultiSurface*. In the conversion LOD 2 is uniformly chosen for the geometries, but it can be changed depending on the use case. This attribute is subsequently set as geometry trait at MultiSurface level through a GeometryPropertySetter transformer. With a second GeometryPropertySetter the geometry name is changed to *lod2MultiSurface*, also at MultiSurface level. The process is designed in a way, that incoming features without any geometries remain untouched. Figure 55 shows this process in FME.

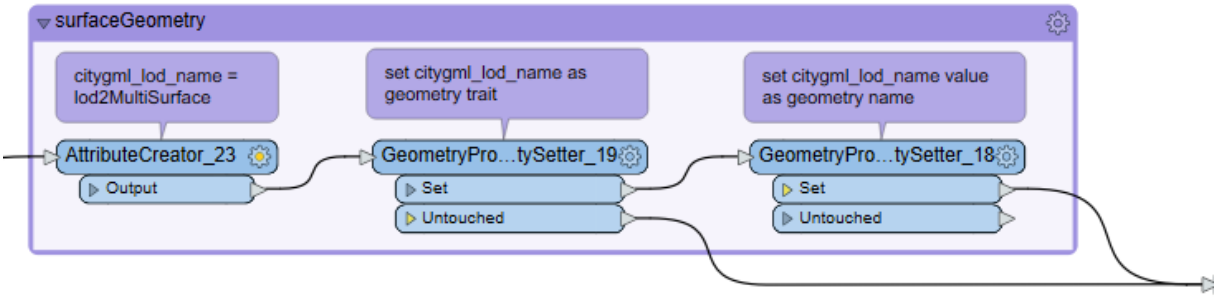


Figure 55: Integrating ADE geometries into CityGML 3.0 in FME.

WeatherData

With mapping the Energy ADE to CityGML 3.0, *WeatherData* is transformed to a feature type. As a consequence, it has its own Writer feature in the FME workspace. The according information is imported together with city objects containing weather data in form of nested list attributes. From them, the *WeatherData* information needs to be extracted, edited and directed to the new Writer. In case of the test data, only *WeatherStation* contains *WeatherData*. Therefore, it is yet only implemented for this feature type. The workflow is depicted in a simplified way in Figure 56.

In a first step, a separate path for the WeatherData is established. Then the parent-child relationship can be implemented by changing the *WeatherStation*'s *gml_id* to *gml_parent_id* and by setting the *citygml_feature_role* to *weatherData*. Because the weather data information is still stored in form of lists per incoming *WeatherStation* feature, they are exploded. Resulting

are individual *WeatherData* features. Their attribute names are subsequently adapted to the required GML encoding by FME and then connected to the Writer.

Due to the new ADE hook mechanism, the parent-child relationship alone is not sufficient to correctly connect the *WeatherData* to its owning class. Therefore, the *gml_ids* of the *WeatherData* features need to be transferred back to *WeatherStation*. Here, they are again aggregated to a list and written to *adeOfAbstractCityObject{}.EnergyADECityObjectProperties.weatherData{}.owns*.

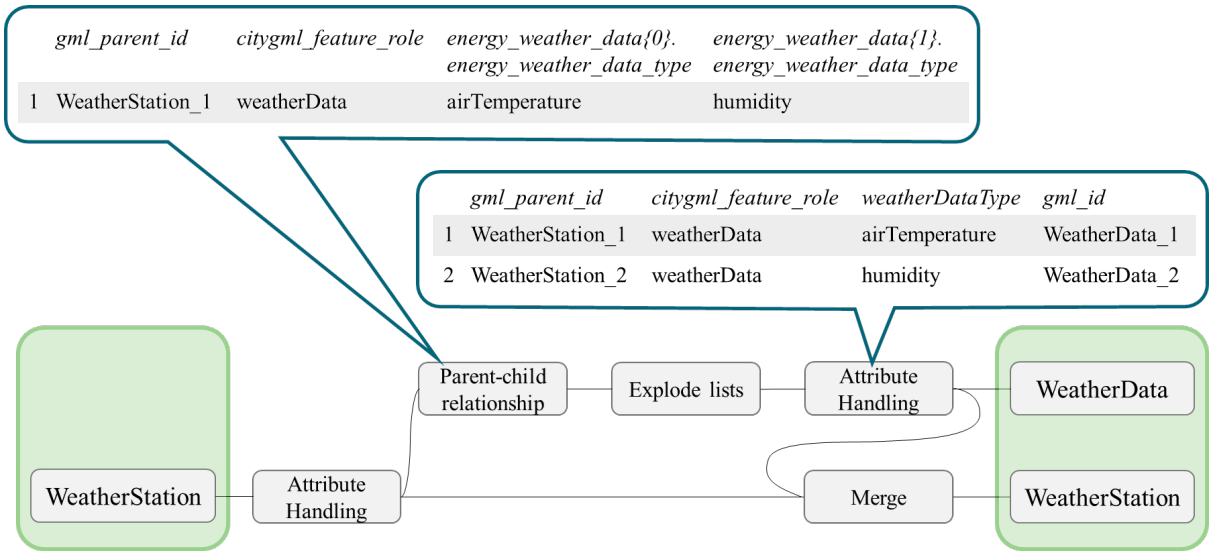


Figure 56: Simplified workflow for extracting and writing WeatherData.

Time Series

Similar to *WeatherData*, the timeseries are mapped to their own feature types. Additionally, they have to be modelled as child elements of *Dynamizer*, which is also newly created. The required timeseries information is also imported with the Reader feature types containing time-depending properties in form of nested attributes. This information is extracted, processed in order to be written to the *Dynamizer* and further passed on for editing to be finally connected to the respective timeseries Writer. The schematic workflow is showed in Figure 57 on the example of *EnergyFlow*. However, the mechanism is the same for all feature types containing time-varying properties.

To be able to eventually write the timeseries data to the correct Writer feature type, it first needs to be determined which type of timeseries each incoming feature describes. Every timeseries must contain the mandatory property *acquisitionMethod* within *variableProperties*. Through a

conditional statement, it is looked up which attribute has an attribute value (e.g. if *energy_energy_amount_energy_**regular_time_series**_energy_variable_properties_energy_time_values_properties_energy_acquisition_method* has a vale; then *RegularTimeSeries*). In case none of the four possible attributes have a value, the timeseries is referenced via XLink. This can be deduced as all time-varying properties are obligatory.

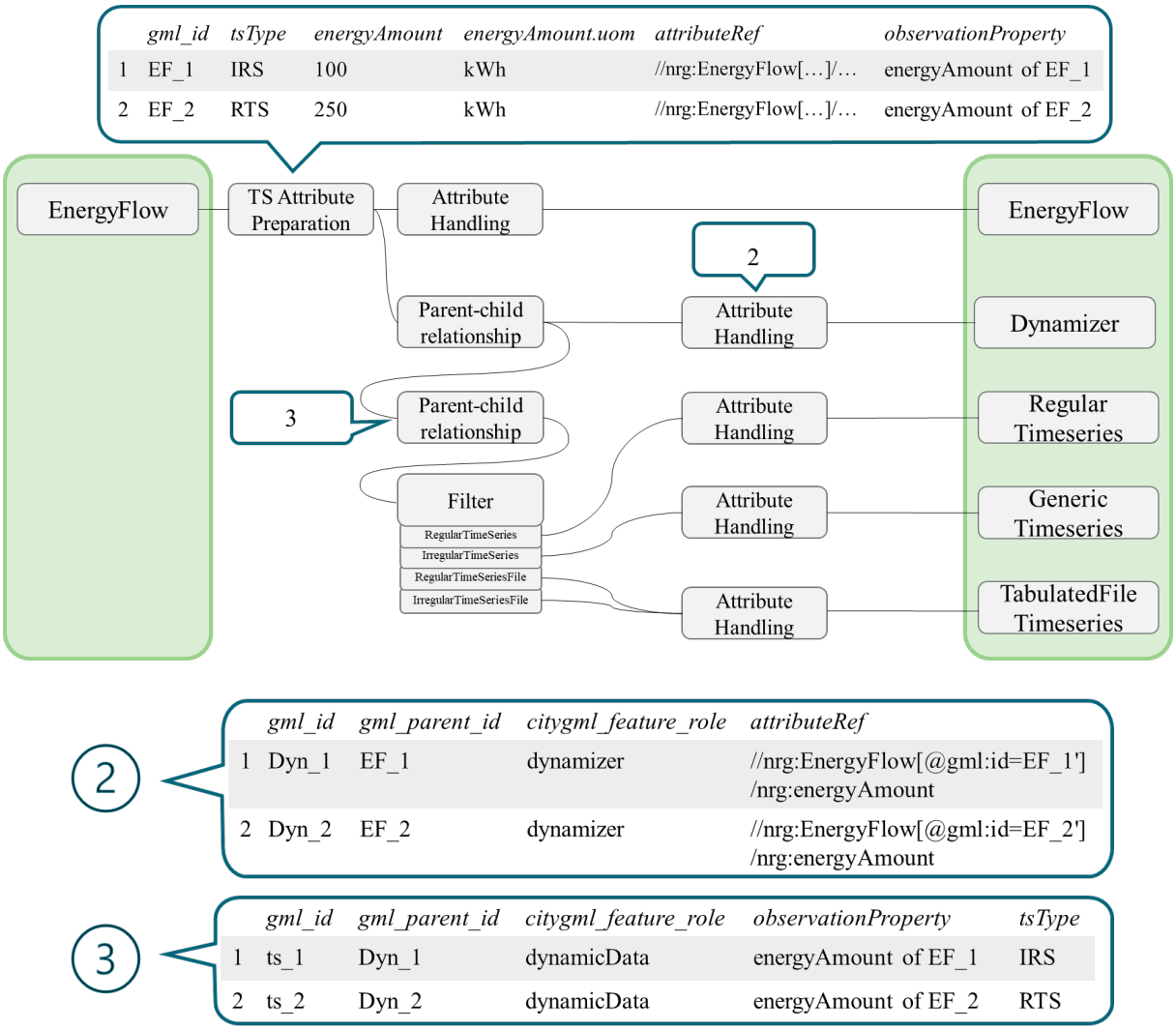


Figure 57: Simplified workflow for extracting and writing timeseries data on the example of EnergyFlow.

Additionally, some attributes are created which are needed for further processing. In the Energy ADE for CityGML 2.0, the time-varying property *energyAmount* was simply described through the timeseries values. But now, it requires its own *Measure* value which is then referenced by the according Dynamizer. Thus, the attributes *energyAmount* and *energyAmount.uom* are created and given input values. Because they have no equivalent in the test data, the values are

currently made up. However, in a more refined version of the Conversion workspace, they could be set through user parameters or be deducted from the timeseries themselves. Beyond this, two attributes are created for *attributeRef* and *observationProperty*. The former requires a XPath to the referenced property. It follows the hierarchy structure in the resulting GML file. In case of *EnergyFlow*, the XPath writes as `//nrg:EnergyFlow[@gml:id='id_energy_flow_1']/nrg:energyAmount`. The latter, *observationProperty*, is rather of descriptive nature.

Followingly, the features are sent to the *Dynamizer* block. In CityGML 3.0, every timeseries object is contained within a *Dynamizer* object which is therefore created first. A parent-child relationship is established through the *gml_parent_id*, which is the *gml_id* of the feature accommodating the time-varying property (here *gml_id* of *EnergyFlow*), and the *citygml_feature_role*, which is always *dynamizer* in this case. Moreover, a *gml_id* for each *Dynamizer* object is created before sending it to the according Writer feature type. Table 2 in Figure 57 summarises this step in a simplified form.

Next, a parent-child relationship between the *Dynamizer* feature and the timeseries feature is developed in the same way. *gml_id* is renamed to *gml_parent_id* and *citygml_feature_role* is set to *dynamicData*. Followingly, the timeseries objects are separated according to the classifying attribute created in the beginning. Depending on the kind of timeseries, the features' attributes are processed to match the required encoding in the Writers.

RegularTimeSeries does not require many changes beyond the attribute renaming. Only the *temporalExtent* is replaced by *firstTimestamp* and *lastTimestamp*.

For the *IrregularTimeSeries* turned *GenericTimeseries*, the attributes are renamed as described earlier. Above this, a new one for *valueType* is created giving it the *TimeseriesTypeValue* double.

The *IrregularTimeSeriesFile* and *RegularTimeSeriesFile* both mapped to *TabulatedFileTimeseries,* are given two new attributes. *valueType* also with the *TimeseriesTypeValue* double, and *fileType*. The latter value is defined through a codelist and is thus simply set to *unknown*. An alternative would be to define a user parameter in FME to individually insert this information. In case of the *RegularTimeSeriesFile,* the attribute *timeColumnName* is given the value *non-existent*.

Schedules

Schedules follow the same pattern as *WeatherData* and the timeseries before. They now all have their own Writer feature types and the according information needs to be extracted from the owning Reader feature type. The workflow for this shows many similarities to the one of the timeseries. It is schematically depicted in Figure 58 for the *Occupants* feature type, although it functions for all feature types containing schedule properties.

In a first step, an attribute is created specifying which kind of schedule id described. This is implemented through conditional statements, testing which of the given attributes carries a value. The attributes used for testing are selected in a way that they are mandatory if they carry this kind of schedule. For example a *ConstantValueSchedule* requires a property value for *averageValue*. Thus, if the corresponding FME encoded attribute (energy_occupancy_rate_energy_constant_value_schedule_energy_**average_value**) contains a value, the feature must describe a *ConstantValueSchedule*.

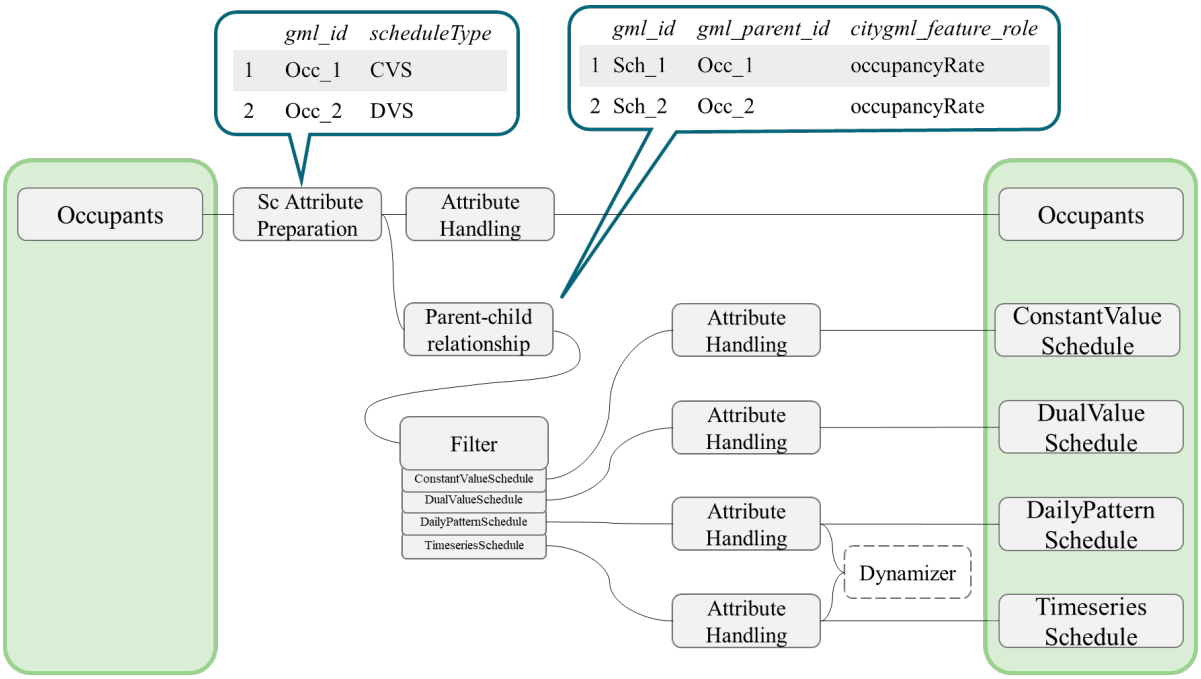


Figure 58: Simplified workflow for extracting and writing schedule data on the example of Occupants.

Then, the features are sent off to establish the parent child relationship between the new schedule feature and the owning feature. As in previous descriptions, this is done through *citygml_feature_role/gml_parent_property* and *gml_parent_id*. Subsequently, the features are filtered on their schedule type attribute and further processed accordingly before connecting them to their respective Writer feature type. Because the *DailyPatternSchedule* and the *TimeseriesSchedule* are further described through timeseries, they are sent to the Dynamizer

block. The *TimeseriesSchedule* are treated in the same way as the remaining time-depending properties. *DailyPatternSchedule* however, is more complex and needs some additional data preparation. Due to the given complexity, this step is adapted to the input data. It is schematically displayed in Figure 59.

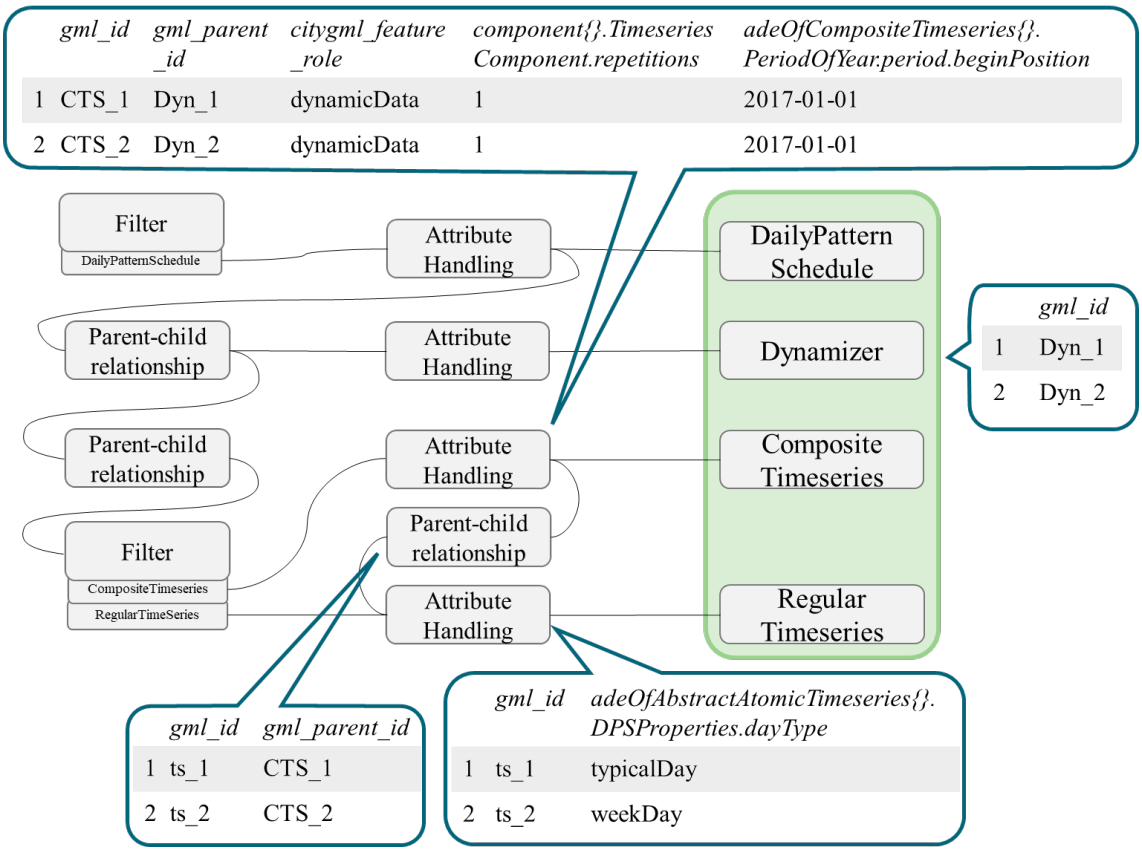


Figure 59: Schematic data preparation for the DailyPatternSchedule consisting of RegularTimeseries.

First, the *DailyPatternSchedule* is filtered and given the sorting attribute value *CompositeTimeseries* for the further processing in the Dynamizer block. After filtering here, the features are prepared for the *CompositeTimeseries* Writer. This includes the creation of the list attribute(s) *component{}.TimeseriesComponent.repetitions* indicating how many components the timeseries has and how often they are repeated. Additionally, the *period* attribute is converted to the ADE hook property of *CompositeTimeseries*. In case there are multiple *PeriodOfYear* given, the attribute is written to the ADE hook of *AbstractAtomicTimeseries* instead. Followingly, the features containing the actual timeseries values are passed on for some further processing to be finally written to the according Writer feature type.

5. Results

This chapter presents the results by summarising the mapping of the classes and by demonstrating how the encoding of a CityGML file enriched with Energy ADE data changes. Additionally, a closer is taken at how the *DailyPatternSchedule* is now encoded through the *CompositeTimeseries*. Furthermore, the efficiency of the mapped Energy ADE is analysed in regard to its size compared to the original Energy ADE for CityGML 2.0.

5.1. Mapped classes

The previous chapter explains in detail how each of the modules and its classes are mapped to CityGML 3.0. Here, the mapping of all classes is summarised in one table, showing how much they were changed and the most important details regarding the changes.

"Obsolete" refers here to the case that the Energy ADE class is replaced by CityGML 3.0, "Adapted" to some larger adjustments and "Mostly taken over" to some minor adjustments to fit the CityGML 3.0 standard.

Table 9: Summary of how much the Energy ADE classes are changed through the mapping to CityGML 3.0.

Module	***Class***	***Status***	***Details***
Core	*_AbstractBuilding / BuildingProperties*	Mostly taken over	Adapted to new hook mechanism, some properties replaced by CityGML 3.0 ones
	AbstractEnergySystem	Adapted	New generalisation class: *AbstractOccupiedSpace*, incorporation in space and geometry concept, property *yearOfManufacture* replaced by CityGML 3.0
	EnergyDemand, WeatherData	Mostly taken over	Adapted to the new hook mechanism, relation to *AbstractDynamizer* to represent time-varying property
Building Physics	*ThermalZone*	Adapted	New generalisation class: *AbstractBuildingSubdivision*, incorporation into space and geometry concept, replacement of properties *floorArea* and *volume* by CityGML 3.0

	ThermalBoundary, ThermalOpening	Adapted	New generalisation class: *AbstractThematicSurface*, incorporation into space and geometry concept, replacement of *area* property
Material and Construction / Layering	*Construction / LayeredMaterial, ReverseConstruction / ReverseLayeredMaterial*	Adapted	Changed name due to new semantic mismatch with CityGML 3.0 conception of construction
	Layer, LayerComponent	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
	AbstractMaterial, Gas, SolidMaterial	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
	ImageTexture	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
Occupant Behaviour	*UsageZone*	Adapted	New generalisation class: *AbstractBuildingSubdivision*, incorporation into space and geometry concept, replacement of property *floorArea* by CityGML 3.0
	BuildingUnit	Adapted	Now extends CityGML 3.0 *BuildingUnit* with additional properties through ADE hook, incorporation into space and geometry concept, replacement of property *floorArea* by CityGML 3.0
	Occupants, Household	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
	Facilities, DHWFacilities, LightingFacilities, ElectricalAppliances	Adapted	New generalisation class: *AbstractOccupiedSpace*, incorporation into space and geometry concept
Energy Systems	*AbstractEnergy ConversionSystem, Boiler, ElectricalResistance, CombinedHeatPower, MechanicalVentilation,*	Mostly taken over	Incorporation into space and geometry concept, generalisation class derives from *AbstractOccupiedSpace*

	AirCompressor, Chiller, GenericConversion System, HeatPump, HeatExchanger, AbstractSolarEnergy System, Photovoltaic System, SolarThermal System, Photovoltaic ThermalSystem		
	AbstractEnergy DistributionSystem, ThermalDistribution System, Power DistributionSystem	Mostly taken over	Incorporation into space and geometry concept, generalisation class derives from *AbstractOccupiedSpace*
	AbstractStorageSystem, ThermalStorageSystem, PowerStorageSystem	Mostly taken over	Incorporation into space and geometry concept
	EmitterSystem	Mostly taken over	Incorporation into space and geometry concept
	EnergyFlow, EnergySource	Mostly taken over	Relation to *AbstractDynamizer* to represent time-varying property
	SystemOperation	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
Time Series	*AbstractTimeSeries*	Obsolete	variableProperties are mapped to *AbstractTimeseries* with the ADE hook
	RegularTimeSeries / RegularTimeseries	Adapted	Incorporated into the CityGML 3.0 dynamizer module as specialisation class of *AbstractAtomicTimeseries*
	IrregularTimeSeries / GenericTimeseries	Obsolete	Replaced by *GenericTimeseries* in the Dynamizer module
	RegularTimeSeriesFile, IrregularTimeSeriesFile /TabulatedFileTimeseries	Obsolete, Adapted	Both classes largely replaced by *TabulatedFileTimeseries* in Dynamizer module, addition of properties *recordSeparator* and *timeInterval* with the ADE hook

Schedules	*AbstractSchedule, ConstantValueSchedule, DualValueSchedule*	Adapted	Changed to stereotype «FeatureType», new way for properties to reference to schedules
	DailyPatternSchedule	Adapted	Changed to stereotype «FeatureType», only one property containing time-depending values, relation to *AbstractDynamizer*, complex timeseries are now covered through *CompositeTimeseries* in the Dynamizer module
	TimeSeriesSchedule / TimeseriesSchedule	Adapted	Changed to stereotype «FeatureType», relation to *AbstractDynamizer*
Weather Data	*WeatherStation*	Adapted	New generalisation class: *AbstractPhysicalSpace*. Incorporation into space and geometry concept

5.2. Comparison of Encodings

The conversion shows that data can successfully be converted to the mapped Energy ADE for CityGML 3.0 without any loss of information. Although the content of the Energy ADE remains the same, the way it is encoded in GML files does change in some parts. This is demonstrated in the following by comparing the test dataset before and after the conversion.

ADE Hook Mechanism

Properties which are added to an existing class with the ADE hook mechanism are in CityGML 2.0 at the same hierarchy level as native CityGML properties (see Encoding 1, line 6).

```
1  <core:cityObjectMember>
2      <bldg:Building gml:id="id_building_01">
3        <gml:description>This is Building 1</gml:description>
4        <gml:name>Snoke's Palace</gml:name>
5        <core:creationDate>2019-11-17</core:creationDate>
6        <energy:buildingType>Terraced House</energy:buildingType>
7        <energy:constructionWeight>medium</energy:constructionWeight>
8        <energy:energyPerformanceCertification>
9            <energy:EnergyPerformanceCertification>
10             <energy:rating>B</energy:rating>
11             <energy:name>CasaClima</energy:name>
```

```
12              <energy:certificationId>CC_12345_AA</energy:certificationId>
13           </energy:EnergyPerformanceCertification>
14       </energy:energyPerformanceCertification>
15       <energy:isLandmarked>false</energy:isLandmarked>
16    </bldg:Building>
17 </core:cityObjectMember>
```

Encoding 1: Encoding of ADE hook properties in CityGML 2.0.

In CityGML 3.0, however, the ADE properties are more nested within the file. The new CityGML property *adeOfAbstractBuilding* (Encoding 2, line 6) contains the ADE *BuildingProperties* (Encoding 2, line 7). The actual properties themselves are then written one hierarchy level further down as seen in line 8.

```
1  <core:cityObjectMember>
2     <bldg:Building gml:id="id_building_01">
3        <gml:description>This is Building 1</gml:description>
4        <gml:name>Snoke's Palace</gml:name>
5        <core:creationDate>2019-11-17T00:00:00</core:creationDate>
6        <bldg:adeOfAbstractBuilding>
7           <nrg:BuildingProperties>
8              <nrg:buildingType>Terraced House</nrg:buildingType>
9              <nrg:constructionWeight>medium</nrg:constructionWeight>
10             <nrg:energyPerformanceCertification>
11                <nrg:EnergyPerformanceCertification>
12                   <nrg:rating>B</nrg:rating>
13                   <nrg:name>CasaClima</nrg:name>
14                   <nrg:certificationId>CC_12345_AA</nrg:certificationId>
15                </nrg:EnergyPerformanceCertification>
16             </nrg:energyPerformanceCertification>
17             <nrg:isLandmarked>false</nrg:isLandmarked>
18          </nrg:BuildingProperties>
19       </bldg:adeOfAbstractBuilding>
20    </bldg:Building>
21 </core:cityObjectMember>
```

Encoding 2: Encoding of ADE hook properties in CityGML 3.0.

This revised hook mechanism has the advantage that several ADE properties from different extensions can be included in one dataset with a clear distinction. On the other hand, it also creates more nested file structures with additional four lines of code per object containing ADE properties.

Timeseries

In the Energy ADE for CityGML 2.0, timeseries data is written directly below the time-depending property. This can be seen on the example of *energyAmount* in line 9 of Encoding

3. Following the properties and complex data types, the encoding has several further hierarchy levels, nesting the information.

```
1  <core:cityObjectMember>
2      <bldg:Building gml:id="id_building_01">
3        <gml:description>This is Building 1</gml:description>
4        <gml:name>Snoke's Palace</gml:name>
5        <core:creationDate>2019-11-17</core:creationDate>
6        <energy:demands>
7            <energy:EnergyDemand gml:id="id_ED_1">
8              <gml:name>Space heating energy demand 1</gml:name>
9              <energy:energyAmount>
10                 <energy:RegularTimeSeries gml:id="id_rts_1">
11                   <energy:variableProperties>
12                       <energy:TimeValuesProperties>
13                         <energy:acquisitionMethod>estimation
                           </energy:acquisitionMethod>
14                         <energy:interpolationType>averageInSucceeding
                           Interval</energy:interpolationType>
15                         <energy:qualityDescription>Quality description
                           text</energy:qualityDescription>
16                         <energy:source>Source text</energy:source>
17                       </energy:TimeValuesProperties>
18                   </energy:variableProperties>
19                   <energy:temporalExtent>
20                       <gml:TimePeriod>
21                         <gml:beginPosition>2017-01-01</gml:beginPosition>
22                         <gml:endPosition>2017-12-31</gml:endPosition>
23                         <gml:duration>P1Y0M0D</gml:duration>
24                       </gml:TimePeriod>
25                   </energy:temporalExtent>
26                   <energy:timeInterval
                     unit="year">0.0833</energy:timeInterval>
27                   <energy:values uom="kWh/month">200 180 160 120 80 0 0 0 40
                     60 10 150</energy:values>
28                 </energy:RegularTimeSeries>
29             </energy:energyAmount>
30             <energy:endUse>spaceHeating</energy:endUse>
31             <energy:energyCarrierType>Natural Gas</energy:energyCarrierType>
32           </energy:EnergyDemand>
33       </energy:demands>
34     </bldg:Building>
35 </core:cityObjectMember>
```

Encoding 3: Encoding of an Energy ADE RegularTimeSeries in CityGML 2.0.

Looking at the same content in CityGML 3.0, the deeper hierarchy levels and longer encoding are obvious (Encoding 4). Partly, this is due to *EnergyDemand* already being contained in the ADE hook structure of *AbstractCityObject* (lines 6-9). Beyond this, another level is added through the required *Dynamizer* which owns the timeseries (lines 12-14). The hook mechanism can furthermore be found again within the timeseries to represent the *TimeValuesProperties* (lines 19-27). Moreover, the two new attributes *attributeRef* and *observationProperty* further elongate the encoding of timeseries data.

When writing *(Ir)RegularTimeSeriesFile* data turned *TabulatedFileTimeseries*, yet another hook is applied which leads to more nesting and additional lines of code to convey the same information.

```
<core:cityObjectMember>
    <bldg:Building gml:id="id_building_01">
       <gml:description>This is Building 1</gml:description>
       <gml:name>Snoke's Palace</gml:name>
       <core:creationDate>2019-11-17</core:creationDate>
       <core:adeOfAbstractCityObject>
           <nrg:EnergyADECityObjectProperties>
              <nrg:demands>
                  <nrg:EnergyDemand gml:id="ED_1">
                     <gml:name>Space heating energy demand 1</gml:name>
                     <nrg:dynamizer>
                         <dyn:Dynamizer gml:id="id_dynamizer_510a">
                            <dyn:attributeRef>//nrg:EnergyDemand[@gml:id=ED_1']
                            /nrg:energyAmount</dyn:attributeRef>
                            <dyn:dynamicData>
                                <nrg:RegularTimeseries gml:id="id_rts_1">
                                   <gml:name>Space heating energy demand
                                   1</gml:name>
                                   <dyn:firstTimestamp>2017-01-
                                   01T00:00:00Z</dyn:firstTimestamp>
                                   <dyn:lastTimestamp>2017-12-
                                   31T00:00:00Z</dyn:lastTimestamp>
                                   <dyn:adeOfAbstractTimeseries>
                                       <nrg:TimeValuesProperties>
                                          <nrg:acquisitionMethod>estimation
                                          </nrg:acquisitionMethod>
                                          <nrg:interpolationType>averageIn
                                          SucceedingInterval
                                          </nrg:interpolationType>
                                          <nrg:qualityDescription>Quality
                                          description text
                                          </nrg:qualityDescription>
                                          <nrg:source>Source text</nrg:source>
                                       </nrg:TimeValuesProperties>
                                   </dyn:adeOfAbstractTimeseries>
                                   <dyn:observationProperty>energy amount
                                   spaceHeating</dyn:observationProperty>
                                   <nrg:timeInterval
                                   unit="year">0.0833</nrg:timeInterval>
                                   <nrg:values uom="kWh/month">200 180 160 120
                                   80 0 0 0 40 60 10 150</nrg:values>
                                </nrg:RegularTimeseries>
                            </dyn:dynamicData>
                         </dyn:Dynamizer>
                     </nrg:dynamizer>
                     <nrg:energyAmount uom="test">110</nrg:energyAmount>
                     <nrg:endUse>spaceHeating</nrg:endUse>
                     <nrg:energyCarrierType>Natural Gas</nrg:energyCarrierType>
                  </nrg:EnergyDemand>
              </nrg:demands>
           </nrg:EnergyADECityObjectProperties>
       </core:adeOfAbstractCityObject>

```

```
43      </bldg:Building>
    </core:cityObjectMember>
```

Encoding 4: Encoding of an Energy ADE RegularTimeseries in CityGML 3.0.

5.3. DailyPatternSchedule

The conversion of the *DailyPatternSchedule* is the most complex one out of all classes. In cases where the test data only has one *PeriodOfYear* and one *DailySchedule*, it works correctly (see Encoding 5). Note that as in the other examples, the converted data is very nested due to its inclusion in the Dynamizer module and the revised ADE hook mechanism.

```
1    <core:cityObjectMember>
2      <bldg:Building gml:id="id_building_03">
3           <gml:description>This is Building 1</gml:description>
4           <gml:name>Snoke's Palace</gml:name>
5           <bldg:adeOfAbstractBuilding>
6             <nrg:BuildingProperties>
7                  <nrg:usageZone>
8                    <nrg:UsageZone gml:id="id_building_3_usage_zone_1">
9                         <gml:description>Single usage zone corresponding to the
                          whole ThermalZone</gml:description>
10                        <gml:name>UsageZone 1 of Building 1</gml:name>
11                        <nrg:coolingSchedule>
12                          <nrg:DailyPatternSchedule
                            gml:id="id_building_3_cooling_schedule_1">
13                               <gml:description>This exemplary cooling schedule
                                 contains a typical day with a timeseries of
                                 Boolean values</gml:description>
14                               <nrg:timeDependingValues>0.5
                                 </nrg:timeDependingValues>
15                               <nrg:dynamizer>
16                                 <dyn:Dynamizer gml:id="id_dynamizer_3">
17                                   <dyn:attributeRef>//nrg: DailyPatternSchedule
                                     [@gml:id='id_building_3_cooling_schedule_1']
                                     /nrg:timeDependingValues</dyn:attributeRef>
18                                     <dyn:dynamicData>
19                                       <dyn:CompositeTimeseries
                                         gml:id="id_CompositeTimeseries_3">
20                                           <dyn:adeOfCompositeTimeseries>
21                                             <nrg:PeriodOfYear>
22                                                  <nrg:period>
23                                                    <gml:TimePeriod>
24                                                         <gml:beginPosition>2017-
                                                           01-01
                                                           </gml:beginPosition>
25                                                         <gml:endPosition>2017-
                                                           12-31</gml:endPosition>
26                                                         <gml:duration>P1Y
                                                           </gml:duration>
27                                                    </gml:TimePeriod>
28                                                  </nrg:period>
29                                             </nrg:PeriodOfYear>
30                                           </dyn:adeOfCompositeTimeseries>
```

```
31                                    <dyn:component>
32                                      <dyn:TimeseriesComponent>
33                                           <dyn:repetitions>1
                                             </dyn:repetitions>
34                                           <dyn:timeseries>
35                                             <nrg:RegularTimeseries
                                               gml:id="id_timeseries_24">
36                                                <dyn:firstTimestamp>
                                                   00:00:00
                                                   </dyn:firstTimestamp>
37                                                <dyn:lastTimestamp>
                                                   23:59:00
                                                   </dyn:lastTimestamp>
38                                                <dyn:adeOfAbstract
                                                   Timeseries>
39                                                  <nrg:Time
                                                     ValuesProperties>
                                                       <nrg:acquisition
40                                                     Method
                                                       >estimation</
                                                       nrg:acquisition
                                                       Method>
                                                      <nrg:interpolation
41                                                     Type>averageIn
                                                       Succeeding
                                                       Interval</
                                                       nrg:interpolation
                                                       Type>
                                                    </nrg:Time
42                                                   ValuesProperties>
                                                  </dyn:adeOfAbstract
43                                                 Timeseries>
                                                  <dyn:observationProperty
44                                                 >time depending values
                                                   of id_building_3_
                                                   cooling_schedule_1</
                                                  dyn:observationProperty>
                                                  <dyn:adeOfAbstract
45                                                 AtomicTimeseries>
                                                    <nrg:DPSProperties>
46                                                       <nrg:dayType>
47                                                       typicalDay</
                                                         nrg:dayType>
                                                    </nrg:DPSProperties>
48                                                </dyn:adeOfAbstract
49                                                 AtomicTimeseries>
                                                  <nrg:timeInterval
50                                                 unit="hour">1</
                                                   nrg:timeInterval>
                                                  <nrg:values
51                                                 uom="Boolean">0 0 0 0
                                                   0 0 1 1 1 1 1 1 1 1 1
                                                   1 1 0 0 0 0 0 0
                                                   </nrg:values>
                                               </nrg:RegularTimeseries>
52                                           </dyn:timeseries>
53                                       </dyn:TimeseriesComponent>
54                                    </dyn:component>
55                                 </dyn:CompositeTimeseries>
```

```
56                                  </dyn:dynamicData>
57                               </dyn:Dynamizer>
58                            </nrg:dynamizer>
59                         </nrg:DailyPatternSchedule>
60                      </nrg:coolingSchedule>
61                   </nrg:UsageZone>
62                </nrg:usageZone>
63             </nrg:BuildingProperties>
64          </bldg:adeOfAbstractBuilding>
65     </bldg:Building>
66  </core:cityObjectMember>
67
```

Encoding 5: Encoding of a simple, correct Energy ADE DailyPatternSchedule in CityGML 3.0.

Nonetheless, more complex *DailyPatternSchedules* with several *PeriodOfYear* or *DailySchedules* pose a problem for FME. In such cases, the *CompositeTimeseries* has several components, where each of them contains one single timeseries. Although the data is prepared in the correct way in FME, the Writer establishes a wrong correspondence. As a result, all timeseries are written within one component (Encoding 6, lines 13-69), the remaining components are written but remain empty (Encoding 6, lines 70-74). This leads to a violation against the XSD schema.

```
1  <nrg:UsageZone gml:id="id_building_1_usage_zone_1">
2    <gml:description>Single usage zone corresponding to the whole
      ThermalZone</gml:description>
3    <gml:name>UsageZone 1 of Building 1</gml:name>
4    <nrg:coolingSchedule>
5        <nrg:DailyPatternSchedule gml:id="id_building_1_cooling_schedule_1">
6          <gml:description>This exemplary cooling schedule contains a typical
            day with a timeseries of Boolean values</gml:description>
7          <nrg:timeDependingValues>0.5</nrg:timeDependingValues>
8          <nrg:dynamizer>
9              <dyn:Dynamizer gml:id="id_dynamizer_1">
10               <dyn:attributeRef>//nrg:DailyPatternSchedule[@gml:id='id_
                  building_1_cooling_schedule_1']/nrg:timeDependingValues
                  </dyn:attributeRef>
11               <dyn:dynamicData>
12                   <dyn:CompositeTimeseries gml:id=
                     "id_CompositeTimeseries_1">
13                     <dyn:component>
14                         <dyn:TimeseriesComponent>
15                           <dyn:repetitions>1</dyn:repetitions>
16                           <dyn:timeseries>
17                               <nrg:RegularTimeseries gml:id=
                                 "id_timeseries_01">
18                                 <dyn:adeOfAbstractAtomicTimeseries>
19                                     <nrg:DPSProperties>
20                                       <nrg:dayType>typicalDay
                                          </nrg:dayType>
21                                       <nrg:period>
22
```

```
23                                          <gml:TimePeriod>
                                              <gml:beginPosition>2017-01-
24                                            01</gml:beginPosition>
                                              <gml:endPosition>2017-06-
25                                            30</gml:endPosition>
                                              <gml:duration>P0Y6M
26                                            </gml:duration>
27                                          </gml:TimePeriod>
28                                      </nrg:period>
29                                  </nrg:DPSProperties>
30                              </dyn:adeOfAbstractAtomicTimeseries>
                                <dyn:firstTimestamp>00:00:00
31                              </dyn:firstTimestamp>
                                <dyn:lastTimestamp>23:59:00
32                              </dyn:lastTimestamp>
33                              <dyn:adeOfAbstractTimeseries>
34                                  <nrg:TimeValuesProperties>
                                      <nrg:acquisitionMethod>estimation
35                                    </nrg:acquisitionMethod>
                                      <nrg:interpolationType>averageIn
                                      SucceedingInterval
36                                    </nrg:interpolationType>
37                                  </nrg:TimeValuesProperties>
38                              </dyn:adeOfAbstractTimeseries>
                                <dyn:observationProperty>time depending
                                values of id_building_1_cooling_
39                              schedule_1</dyn:observationProperty>
                                <nrg:timeInterval
40                              unit="hour">1</nrg:timeInterval>
                                <nrg:values uom="Boolean">0 0 0 0 0 0 1 1 1
41                              1 1 1 1 1 1 1 1 1 0 0 0 0 0 0</nrg:values>
42                          </nrg:RegularTimeseries>
                            <nrg:RegularTimeseries gml:id=
43                           "id_timeseries_02">
44                              <dyn:adeOfAbstractAtomicTimeseries>
45                                  <nrg:DPSProperties>
                                      <nrg:dayType>typicalDay
46                                    </nrg:dayType>
47                                    <nrg:period>
48                                      <gml:TimePeriod>
                                          <gml:beginPosition>2017-07-
49                                        01</gml:beginPosition>
                                          <gml:endPosition>2017-12-
50                                        31</gml:endPosition>
                                          <gml:duration>P0Y6M
51                                        </gml:duration>
52                                      </gml:TimePeriod>
53                                    </nrg:period>
54                                  </nrg:DPSProperties>
55                              </dyn:adeOfAbstractAtomicTimeseries>
                                <dyn:firstTimestamp>00:00:00
56                              </dyn:firstTimestamp>
                                <dyn:lastTimestamp>23:59:00
57                              </dyn:lastTimestamp>
58                              <dyn:adeOfAbstractTimeseries>
59                                  <nrg:TimeValuesProperties>
                                      <nrg:acquisitionMethod>estimation
60                                    </nrg:acquisitionMethod>
                                      <nrg:interpolationType>averageIn
```

```
                                        SucceedingInterval
61                                      </nrg:interpolationType>
62                                    </nrg:TimeValuesProperties>
63                                  </dyn:adeOfAbstractTimeseries>
                                    <dyn:observationProperty>time depending
                                    values of id_building_1_cooling
64                                  _schedule_1</dyn:observationProperty>
                                    <nrg:timeInterval
65                                  unit="hour">1</nrg:timeInterval>
                                    <nrg:values uom="Boolean">1 0 1 0 1 0 1 0 1
66                                  0 1 0 1 0 1 0 1 0 1 0 1 0 1 0</nrg:values>
67                                </nrg:RegularTimeseries>
68                              </dyn:timeseries>
69                            </dyn:TimeseriesComponent>
70                        </dyn:component>
71                        <dyn:component>
72                            <dyn:TimeseriesComponent>
73                              <dyn:repetitions>1</dyn:repetitions>
74                            </dyn:TimeseriesComponent>
75                        </dyn:component>
76                    </dyn:CompositeTimeseries>
77                </dyn:dynamicData>
78              </dyn:Dynamizer>
79          </nrg:dynamizer>
80        </nrg:DailyPatternSchedule>
81    </nrg:coolingSchedule>
    </nrg:UsageZone>
```

Encoding 6: Incorrect encoding of a more complex Energy ADE DailyPatternSchedule in CityGML 3.0. The CompositeTimeseries has two components, whereas both timeseries are written within the first one.

However, this is an error produced by the FME GML Writer and not the developed data model itself, which can be proven in two ways. First, a simple CityGML 3.0 test data set without the Energy ADE, containing one *CompositeTimeseries* with several components, is imported into FME with a GML Reader. The validation at this point is successful. Connecting the Reader feature types directly with their corresponding GML Writer feature types and exporting the data, produces the same validation errors as in the conversion workspace. The second way of proving this is by manually altering the converted test data set (the output created by the workspace described in 4.3.2) in a way that the individual timeseries are correctly corresponded to their component. By this, the content itself is not changed, only its position within the GML file (see Encoding 7). Reading in and validating this dataset with a GML Reader produces no errors.

```
1   <nrg:UsageZone gml:id="id_building_1_usage_zone_1">
2     <gml:description>Single usage zone corresponding to the whole
       ThermalZone</gml:description>
3     <gml:name>UsageZone 1 of Building 1</gml:name>
4     <nrg:coolingSchedule>
5         <nrg:DailyPatternSchedule gml:id="id_building_1_cooling_schedule_1">
6           <gml:description>This exemplary cooling schedule contains a typical
```

```
7           day with a timeseries of Boolean values</gml:description>
8          <nrg:timeDependingValues>0.5</nrg:timeDependingValues>
9          <nrg:dynamizer>
10             <dyn:Dynamizer gml:id="id_dynamizer_1">
                 <dyn:attributeRef>//nrg:DailyPatternSchedule[@gml:id='id_
                  building_1_cooling_schedule_1']/nrg:timeDependingValues
11                </dyn:attributeRef>
12                <dyn:dynamicData>
                      <dyn:CompositeTimeseries gml:id=
13                    "id_CompositeTimeseries_1">
14                      <dyn:component>
15                          <dyn:TimeseriesComponent>
16                            <dyn:repetitions>1</dyn:repetitions>
17                            <dyn:timeseries>
                                  <nrg:RegularTimeseries gml:id=
18                                "id_timeseries_01">
19                                  <dyn:adeOfAbstractAtomicTimeseries>
20                                      <nrg:DPSProperties>
                                          <nrg:dayType>typicalDay
21                                         </nrg:dayType>
22                                        <nrg:period>
23                                            <gml:TimePeriod>
                                                <gml:beginPosition>2017-01-
24                                              01</gml:beginPosition>
                                                <gml:endPosition>2017-06-
25                                              30</gml:endPosition>
                                                <gml:duration>P0Y6M
26                                              </gml:duration>
27                                            </gml:TimePeriod>
28                                        </nrg:period>
29                                      </nrg:DPSProperties>
30                                  </dyn:adeOfAbstractAtomicTimeseries>
                                    <dyn:firstTimestamp>00:00:00
31                                  </dyn:firstTimestamp>
                                    <dyn:lastTimestamp>23:59:00
32                                  </dyn:lastTimestamp>
33                                  <dyn:adeOfAbstractTimeseries>
34                                      <nrg:TimeValuesProperties>
                                          <nrg:acquisitionMethod>estimation
35                                         </nrg:acquisitionMethod>
                                          <nrg:interpolationType>averageIn
                                           SucceedingInterval
36                                         </nrg:interpolationType>
37                                      </nrg:TimeValuesProperties>
38                                  </dyn:adeOfAbstractTimeseries>
                                    <dyn:observationProperty>time depending
                                    values of id_building_1_cooling_
39                                  schedule_1</dyn:observationProperty>
                                    <nrg:timeInterval
40                                  unit="hour">1</nrg:timeInterval>
                                    <nrg:values uom="Boolean">0 0 0 0 0 0 1 1 1
41                                  1 1 1 1 1 1 1 1 0 0 0 0 0 0</nrg:values>
42                                </nrg:RegularTimeseries>
43                            </dyn:timeseries>
44                          </dyn:TimeseriesComponent>
45                      </dyn:component>
46                      <dyn:component>
47                          <dyn:TimeseriesComponent>
48                            <dyn:repetitions>1</dyn:repetitions>
```

```
                                <dyn:timeseries>
                                    <nrg:RegularTimeseries
                                     gml:id="id_timeseries_02">
                                      <dyn:adeOfAbstractAtomicTimeseries>
                                          <nrg:DPSProperties>
                                            <nrg:dayType>typicalDay
                                             </nrg:dayType>
                                            <nrg:period>
                                                <gml:TimePeriod>
                                                  <gml:beginPosition>2017-07-
                                                  01</gml:beginPosition>
                                                  <gml:endPosition>2017-12-
                                                  31</gml:endPosition>
                                                  <gml:duration>P0Y6M
                                                  </gml:duration>
                                                </gml:TimePeriod>
                                            </nrg:period>
                                          </nrg:DPSProperties>
                                      </dyn:adeOfAbstractAtomicTimeseries>
                                      <dyn:firstTimestamp>00:00:00
                                      </dyn:firstTimestamp>
                                      <dyn:lastTimestamp>23:59:00
                                      </dyn:lastTimestamp>
                                      <dyn:adeOfAbstractTimeseries>
                                          <nrg:TimeValuesProperties>
                                            <nrg:acquisitionMethod>estimation
                                             </nrg:acquisitionMethod>
                                            <nrg:interpolationType>averageIn
                                             SucceedingInterval
                                             </nrg:interpolationType>
                                          </nrg:TimeValuesProperties>
                                      </dyn:adeOfAbstractTimeseries>
                                      <dyn:observationProperty>time depending
                                      values of id_building_1_cooling_
                                      schedule_1</dyn:observationProperty>
                                      <nrg:timeInterval unit="hour">1
                                      </nrg:timeInterval>
                                      <nrg:values uom="Boolean">1 0 1 0 1 0 1 0 1
                                      0 1 0 1 0 1 0 1 0 1 0 1 0</nrg:values>
                                    </nrg:RegularTimeseries>
                                </dyn:timeseries>
                            </dyn:TimeseriesComponent>
                        </dyn:component>
                    </dyn:CompositeTimeseries>
                </dyn:dynamicData>
            </dyn:Dynamizer>
        </nrg:dynamizer>
    </nrg:DailyPatternSchedule>
  </nrg:coolingSchedule>
</nrg:UsageZone>
```

Encoding 7: Manually corrected encoding of a more complex Energy ADE DailyPatternSchedule in CityGML 3.0. Each RegularTimeseries is written within one component.

5.4. File size comparison

As it can be anticipated through the examples above, the file size for a dataset in terms of number of lines increases with the Energy ADE for CityGML 3.0. Table 10 shows this on the example of the used test dataset where its number of lines is determined with and without the Energy ADE, both for CityGML 2.0 and CityGML 3.0. In all scenarios, the size increases with the new standard. However, including the Energy ADE (+12%) more than without it (+4.5%). This also becomes evident by looking at the increased overhead produced by the Energy ADE (how many lines in the file belong to the ADE).

Table 10: File size comparison for the test dataset. The values refer to the number of lines in each scenario.

	Without Energy ADE	*With Energy ADE*	*Overhead Energy ADE*
CityGML 2.0	1895	15090	13195
CityGML 3.0	1980	16911	14931
	+4.5%	+12%	+13%

The reason behind is presumably the frequent use of the ADE hook mechanism, which requires in its revised version more lines to encode the same information. Moreover, the mapping of some Energy ADE classes to CityGML 3.0 classes (e.g. *BuildingUnit, RegularTimeSeriesFile, DailyPatternSchedule*) requires the additional application of the hook mechanism to include all given properties. Additionally, the timeseries now also require more space in the encoding due to the adaption to the Dynamizer module.

Nevertheless, if file size is a limiting factor for an application, CityGML files can effectively be compressed as zip or gzip files. These zipped files can then also be read with FME.

Concerning the XSD schema files, the Energy ADE for CityGML 2.0 one comprises 2452 lines, whereas the mapped one for CityGML 3.0 only 1663. This is also reflected by the file size in terms of storage, with the mapped schema almost requiring half the space (132KB vs 73KB). The reason behind is that many ADE properties got replaced by CityGML ones, as well as some of the classes. Nevertheless, the same amount of information can be conveyed with the updated version of the Energy ADE. Thus, the size of the schema file does not necessarily reflect the size of the data carrying the actual information.

6. Discussion

This chapter sets the generated Energy ADE for CityGML 3.0 and the corresponding conversion in a broader context and discusses certain decisions, outcomes and possible alternatives.

6.1. A unique solution?

The data conversion proves the successful mapping of the Energy ADE 1.0 to CityGML 3.0 without any loss of information. However, it is also clear from the given explanations in chapter 4.1, that there are several possibilities to conduct the mapping. The implemented one follows the guiding principle of a mapping on a logical and conceptual level.

Minimum Mapping

One alternative viable solution would be a "minimum mapping". In this scenario, the only adjustments are the ones strictly necessary for the ADE to work with CityGML 3.0. This includes the adaption to the new ADE hook mechanism in the case of the additional *AbstractBuilding* properties, *WeatherData* and *EnergyDemand*. Furthermore, the names of the generalisation classes would need to be updated, e.g. from *_CityObject* in CityGML 2.0 to *AbstractCityObject* in CityGML 3.0. Lastly, the stereotype «type» in *WeatherData*, *ServiceLife*, the schedules and timeseries would need to be changed to a reasonable alternative.

The ADE classes would remain at the same parent classes as before. For example *AbstractThermalZone* and *AbstractUsageZone* would derive from *AbstractCityObject*. *AbstractLayeredMaterial* (renaming would also in this scenario be necessary) and all its related classes would remain subclasses of *AbstractFeature*.

An open question is how the timeseries would be handled in this scenario. One option would be to still make use of the Dynamizer module and its adjacent classes. Although a more minimalist approach would be to also convert them to «FeatureType» and derive *AbstractTimeSeries* from *AbstractFeature*. The references to *AbstractTimeSeries* could then be implemented in the same way as it is done in this thesis' solution for the schedules.

As a result, the mapped ADE would be very similar to the original Energy ADE for CityGML 2.0, also without any loss of information. Compared to the applied implementation in this thesis,

the minimum mapping approach is less complex and keeps the ADE classes on a higher level within the UML class diagrams.

On the other hand, it disregards any of the changes introduced in CityGML 3.0. The new space and geometry concept would not apply as nothing is derived below *AbstractCityObject*. This misses out on the additional semantics provided and furthermore requires the Energy ADE to keep explicitly define its geometries. Moreover, the version history could not be consistently implemented due to the classes deriving from *AbstractFeature*. Finally, almost none of the properties, such as *volume* in *ThermalZone* or *area* in *ThermalBoundary*, could be replaced by CityGML 3.0 properties.

Middle Ground

A second alternative to map the Energy ADE to CityGML 3.0 is by finding a middle ground between the applied "integrate as much as possible" solution and the minimum mapping.

For this, ADE objects would be integrated into the space and geometry concept, where the contextual relation is obvious. However, only the abstract space classes would be considered. This would for example be the case for *AbstractThermalZone* which then derives either from *AbstractSpace* or *AbstractLogicalSpace*.

Classes which derive in the Energy ADE for CityGML 2.0 from *_CityObject*, but do not carry any geometries would remain being subclassed from *AbstractCityObject* (e.g. *Facilities* and *AbstractEnergySystem*). Furthermore, it could be discussed whether to derive the remaining classes from *AbstractFeature* or *AbstractFeatureWithLifespan*.

This mapping strategy would allow to make use of the geometry definitions in CityGML 3.0. At the same time, features without geometries in the Energy ADE for CityGML 2.0 would also not be given any. Thus, the middle ground solution accounts for some of the changes in CityGML 3.0 while at the same time keeping the mapped Energy ADE relatively similar to the original one.

Both of the presented alternatives are most likely implementable without any loss of information. Only the level of integration into CityGML 3.0 changes, and with this how much additional context is given through it.

However, compared to the options above, the applied mapping strategy fully accounts for all changes and additional features introduced with CityGML 3.0. It follows the in the CityGML 3.0 Conceptual Model standard proposed strategy to derive classes where they best semantically fit. Additionally, it adheres to the CityGML 3.0 developers' ideal that nothing should be derived from *AbstractCityObject* itself anymore. Because if something is a city object, and thus physically experienceable, it should also be integrated into the space and geometry concept.

Eventually, only applications and testing of the three mapping solutions can show which of them is the most practical.

6.2. Geometry representations

As a result of deploying the general mapping principles, in particular the first one – integrate as much as possible, all ADE classes formerly deriving from *_CityObject* are now integrated into the space and geometry concept of CityGML 3.0. This comes with several benefits.

To begin with, geometries no longer have to be explicitly defined within the ADE. Instead, the CityGML 3.0 geometries are made use of. Additionally, several new classes implementing the space concept are introduced with the new OGC standard. They thus increase the amount of possible parent classes and furthermore add another layer of semantic meaning to their deriving ADE classes. Consequently, the Energy ADE integrates seamlessly into the designed logic of CityGML 3.0.

Yet, some new challenges and open questions arise through this mapping. In the Energy ADE for CityGML 2.0, only one explicit geometry representation is foreseen per feature. Now, it is possible to model multiple geometries in various LODs for the features. Though, it is not particularly defined what each LOD of an ADE class should represent. For example to specify how a *ThermalZone* is modelled in LOD 2 versus in LOD 3. Generally, the common LOD notion as defined in CityGML 3.0 should be applied and adapted to the use case. Moreover, it can be specified which geometry representations are allowed for a class depending on its context. To stay with the example of *ThermalZone*, it would be possible to stipulate that it can only be bound by *ThermalBoundary*. Furthermore, classes can be given a maximum LOD in which they are allowed to be modelled or restrict the geometry representation all over. For instance, it can be considered to represent *Facilities* only through a *lod0Point* geometry as more detail would not add any further value to an energy simulation. However, these detailed

geometry definitions and considerations are out of scope for this work and subject to further testing in practice.

Another consequence of the applied mapping approach is that some ADE classes now have the possibility to be geometrically represented as opposed to before in the Energy ADE for CityGML 2.0. This includes *BuildingUnit*, now integrated into the CityGML 3.0 *BuildingUnit* feature type, *AbstractEnergySystem* and *Facilities* both being mapped to *AbstractOccupiedSpace*. Before, they all derived from *_CityObject* without having any geometry.

Seemingly, this contradicts the aim of this work to map the Energy ADE without any changes of its content and functionalities. While this is true to some extent, the overall logic and consistent mapping outweighs attaining this aim. The decision for this also goes back to a remark from T.H. Kolbe, one of CityGML 3.0's main developers. According to him, nothing should derive from *AbstractCityObject* itself anymore. Instead, all city objects are supposed to be mapped to one of the abstract space classes (see Figure 21). The resulting issue of the extended functionality could be circumvented by restricting the geometric modelling of those classes as described before. This way, the CityGML 3.0 modelling style would be respected without extending the Energy ADE's functionalities.

A last consideration regarding the geometries in the Energy ADE for CityGML 3.0 is the actual modelling of valid geometries. For *ThermalZone*, this regards for instance the specification of the normal direction (inwards or outwards) or the watertight boundary representation through *ThermalBoundary*. Furthermore, it could be defined that a *ThermalBoundary* should have a cavity at the place of a *ThermalOpening* to prevent overlayed surfaces. However, these specifications are also out of scope of this thesis and are meant to represent a chance for further reasoning on the Energy ADE and CityGML 3.0. In the meantime, general definitions of valid geometries as they are described in chapter 9 of Arroyo Ohori et al., 2022 apply in addition to the specifications in the CityGML 3.0 conceptual model standard (Kolbe et al., 2021) and regarding explanations in Agugiaro et al., 2018.

6.3. Considerations beyond mapping

Throughout the explanations in chapter 4.1 it becomes clear that, even with rather rigidly set mapping principles, are several possibilities to implement something. In this context, also some options come up which would go beyond a sole mapping and extend the Energy ADE's

functionalities. As this is out of scope, they are eventually not implemented. Nevertheless, these options are again highlighted at this point.

The to the Dynamizer module added class *RegularTimeseries*, only accepts numeric values for the data type MeasureList (property *values*). Its counterpart, *GenericTimeseries* however, accepts several other data types such strings, geometry object or booleans. Thus, the *RegularTimeseries* could also be extended with these types to be more coherent with CityGML 3.0.

Furthermore, the requirement for time-varying properties to have a connection to the *Dynamizer* could be loosened a bit. By this, it is also possible to represent the property through just one value instead of a necessary timeseries. The single value can for example be an average of the other values. This slightly extended conception does not take away any of the ADE's functionality, but additionally allows for an alternative simpler modelling.

Lastly, a relation from *AbstractCityObject* to *AbstractLayeredMaterial* could be added. The relation would enable every city object, not only buildings and thermal zones, to further specify the materials it is made of. As such, the additional functionality would be similar in its purpose to the corresponding relation to *EnergyDemand*.

These few examples show that possibly not all benefits of CityGML 3.0 can be made use of by a mapping of the Energy ADE with the aim to not change any of its functionalities. The reason behind is that the Energy ADE 1.0 was originally developed for CityGML 2.0. A sole mapping can therefore not account for the changes within the OGC standard itself. Consequently, if the Energy ADE was newly designed for CityGML 3.0, the result would most likely be different to the one obtained in this thesis.

As a side note, there are also two suggestions for future CityGML 3.0 releases. First is the inclusion of a class to represent regular timeseries in the Dynamizer module. Those are frequently used and are more space efficient in the encoding compared to representing the same information through GenericTimeseries. Second, also in the Dynamizer module, is the addition of a record separator property to *TabulatedFileTimeseries*. It is usually needed for a complete reading of the time-varying information in a file, especially when working with different file formats.

6.4. Data conversion

The developed conversion tool in FME transfers the test data to Energy ADE for CityGML 3.0 data without losing any content, even though some *DailyPatternSchedules* require manual postprocessing.

But beyond that, the workspace also has some limitations. First of all, it is comparatively large and slow to edit. This is mostly due to the very long nested attributes which are each individually renamed. The renaming process itself is furthermore rather repetitive. Each feature type at least needs one transformer to rename the attributes accordingly. Depending on the number of attributes, the transformer can get very rich in information.

Therefore, a solution to rename all attributes of all features automatically with the same transformers would be ideal. A possible approach is through the BulkAttributeRenamer, which renames all incoming attributes according to predefined rules. However, those rules are difficult to determine in a way that they fit all attributes. Especially when it comes to the capitalisation of the letters as this is not rule based, but solely dependent on how the names are written in the UML diagram. Another option could be to already import the Energy ADE for CityGML 2.0 data with a GML Reader instead of a CityGML one. This could prevent the extensive renaming of attributes overall. But this option is yet to be explored. Beyond this, those issues likely solve themselves once a CityGML 3.0 Reader and Writer is available in FME.

Although the applied attribute renaming process is tedious, it works without any problems in regard of the generated output. Beyond this, the workspace contains another limitation through its partly adaption to the input test data and some hardcoded information. For instance, the *WeatherData* is only retrieved from *WeatherStation*, whereas it can be contained within all city objects. This can be solved by creating a *WeatherStation* block, similar to the ones for timeseries and schedules, where each city object send its features to. Furthermore, the LOD when integrating the ADE geometries to CityGML 3.0 geometries, is uniformly set to LOD 2. Similarly, the newly required static attribute values for the time-depending properties are hardcoded. Both of those problems could be solved by applying user parameters. They can be set by the user before the workspace is run.

7. Conclusion

This thesis proposes a mapping of the Energy ADE 1.0 to the recently published CityGML 3.0 standard without losing any of the information conveyed. In order to do so, it follows a model-driven approach adapted from van den Brink et al., 2013. It includes the creation of UML class diagrams, implemented in Enterprise Architect, and the automatic derivation of a corresponding XSD schema file through ShapeChange. Additionally, sample data was created and converted with FME to CityGML 3.0 plus Energy ADE to test and verify the mapped extension. Eventually, the research question including the sub-questions can be answered as follows.

7.1. With regard to the Research Objective

How and to what extent need the Energy ADE for CityGML 2.0 be adapted to be conformant with the newly released CityGML 3.0 standard?

Overall, the ADE hook mechanism and the correspondences of the Energy ADE classes to CityGML 3.0 have successfully been adapted. This is implemented in a manner which accounts for the introduced changes in CityGML 3.0, by making use of the space and geometry concept, the versioning possibilities as well as the provided structures to model time-dependent data. Beforehand defined rules, such as to integrate the Energy ADE classes as much as possible and to maintain a logical symmetry or to prefer *AbstractFeatureWithLifespan* over *AbstractFeature* as a generalisation class, ensure a consistent mapping throughout all modules. Another result of the applied strategy is that several Energy ADE properties could be replaced by CityGML 3.0 ones.

Which classes of the Energy ADE 1.0 become obsolete, which ones need to be adapted and which ones can mostly be taken over?

Out of the 61 classes of the Energy ADE, 4 got obsolete through the mapping, 21 were adapted and 36 were mostly taken over. “Obsolete” refers here to the case that the Energy ADE class is replaced by CityGML 3.0, “Adapted” to some larger adjustments and “Mostly taken over” to some minor adjustments to fit the CityGML 3.0 standard. The table below depicts a small excerpt of the mapping results, showing how much the classes have changed and the most important details regarding the changes. A complete version of the table can be found in Chapter 5.1.

Table 11: Details of how selected Energy ADE classes have been changed while mapping to CityGML 3.0. Please refer to Table 9 for the full version.

Class	***Status***	***Details***
AbstractEnergySystem	Adapted	New generalisation class: *AbstractOccupiedSpace*, incorporation in space and geometry concept, property *yearOfManufacture* replaced by CityGML 3.0
ThermalZone	Adapted	New generalisation class: *AbstractBuildingSubdivision*, incorporation into space and geometry concept, replacement of properties *floorArea* and *volume* by CityGML 3.0
Occupants	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
AbstractEnergyConversionSystem	Mostly taken over	Incorporation into space and geometry concept, generalisation class derives from *AbstractOccupiedSpace*
IrregularTimeSeries	Obsolete	Replaced by *GenericTimeseries* in the Dynamizer module

What will the Energy ADE data model for CityGML 3.0 look like, both in terms of UML encoding and XSD file?

This is shown on the example of the class *AbstractEnergySystem* in the Core module. Its parent class has changed through the mapping from *_CityObject* to *AbstractOccupiedSpace* which is also reflected in the UML class diagram (Figure 60) and the XSD schema file (Figure 61). The complete UML diagrams can be found in Appendix A: UML diagrams of Energy ADE for CityGML 3.0. The XSD schema file can be accessed over the Git repository.

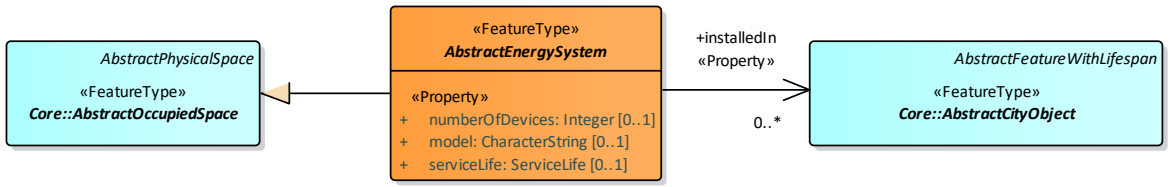


Figure 60: Excerpt of the Energy ADE for CityGML 3.0 Core module, showing the class AbstractEnergySystem.

```
<element abstract="true" name="AbstractEnergySystem" substitutionGroup="core:AbstractOccupiedSpace" type=
"nrg:AbstractEnergySystemType"/>
<complexType abstract="true" name="AbstractEnergySystemType">
  <complexContent>
    <extension base="core:AbstractOccupiedSpaceType">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="installedIn" type=
        "core:AbstractCityObjectPropertyType"/>
        <element minOccurs="0" name="numberOfDevices" type="integer"/>
        <element minOccurs="0" name="model" type="string"/>
        <element minOccurs="0" name="serviceLife">
          <complexType>
            <complexContent>
              <extension base="gml:AbstractMemberType">
                <sequence minOccurs="0">
                  <element ref="nrg:ServiceLife"/>
                </sequence>
                <attributeGroup ref="gml:AssociationAttributeGroup"/>
              </extension>
            </complexContent>
          </complexType>
        </element>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 61: Excerpt of the Energy ADE for CityGML 3.0 XSD schema file for the class AbstractEnergySystem.

How can Energy ADE for CityGML 2.0 data be converted to Energy ADE for CityGML 3.0 data?

At the moment, and within the scope of this thesis, data can be converted with the software FME. It is imported with a CityGML Reader and exported with a GML Writer providing the corresponding XSD schema files. In between, mainly the attributes are renamed accordingly. Furthermore, the Energy ADE geometries are transformed into standard CityGML 3.0 ones. Lastly, the timeseries and schedules which are now individual feature types, are extracted from the imported data, manipulated and connected to their own Writer. This process works for almost every feature without any loss of information. The only exception to this are *DailyPatternSchedules* which are mapped to *CompositeTimeseries* with several *TimeseriesComponents*. In this case, the GML Writer does not interpret and export the data correctly. Yet, this is a software limitation (or bug) and not an error of the mapped Energy ADE itself. The proof for that is that it can be solved through a simple manual postprocessing as seen in Chapter 5.3.

7.2. Open Issues and Future Work

The applied mapping strategy leads in parts to an extended functionality of the Energy ADE. This mainly affects the wider possibilities to geometrically represent city objects through the incorporation into the geometry concept of CityGML 3.0. Thus, the specification of allowed geometry representations and LODs for each ADE class deriving from *AbstractCityObject* calls for further reasoning to attain a complete definition of the Energy ADE for CityGML 3.0. A further refinement of the work could be to create individual packages for the modules as it is done in the Energy ADE for CityGML 2.0. Currently, all UML class diagrams are modelled within one package, inhibiting the creation of package-based diagrams.

Moreover, alternative mapping solutions could be implemented. One of them is the minimum mapping approach, aligning the Energy ADE only as much as absolutely necessary to comply with the new standard. However, this approach would not take the changes and with this, the potential of CityGML 3.0 into account. Either way, only testing and real-life use cases can show the most practical solution.

Furthermore, due to the novelty of the Energy ADE for CityGML 3.0, it only fulfils one of its two original purposes. It stores UBEM relevant data in a standardised way and thus enhances data exchange and interoperability. But at the moment, it cannot serve as input data or format for simulations until corresponding software implements its support.

7.3. Outlook

All over, this work allows me to provide some insights for future versions of the Energy ADE which are then likely natively modelled on CityGML 3.0. This accounts especially for the handling of time-varying properties as well as considerations regarding the use of the newly provided space concept. Beyond that, I am aspiring to contribute with this work to the establishment of the new CityGML 3.0 standard. For other ADEs which also need to be adapted to function with it, this thesis can provide an example on methodology and on how encountered issues are solved.

In addition, the current energy crisis in Europe with increasing prices and the question of long-term energy security makes this topic even more relevant. This includes the rising costs for building materials such as cement, which is highly energy consuming in its production. Therefore, renovation of the already existing building stock is crucial, also from an economic

point of view. For the future, it is hence inevitable to create a sustainable, energy efficient building stock.

Thus, it should be invested in research and development to support this goal. Urban Building Energy Modelling can hereby play an important role, to help efficiently allocating resources where they have the most impact. With this thesis, I believe I am contributing to this bigger goal by working towards the further development and establishment of UBEM.

7.4. Personal Reflection

The Master programme Geomatics at the Delft University of Technology enabled me to conduct this research through its interdisciplinary nature and technical depth. Many of the completed courses prepared me directly or indirectly for this big project. This includes the acquisitions of skills in software and programming languages, the knowledge in the domain of 3D modelling or valuable soft skills such as fast familiarisation with new topics and problems.

Courses of special relevance were "Geographical Information Systems (GIS) and Cartography", giving me my first insights into the world of FME, "Geo Database Management Systems", showing me the complexity of UML modelling, "3D Modelling of the Built Environment", providing me with knowledge of 3D city models and geometries, and "Geo-information Governance" teaching me the dos and don'ts of scientific writing. Beyond this, the elective "Geomatics as support for energy applications" introduced me into the world of CityGML and UBEM and started my interest in this specific research area. It furthermore enabled me to refine my skills in UML modelling and the software FME. This also prepared me greatly for my internship at con terra, where I was able to put this into practice. It was also where I first got in touch with CityGML 3.0. The new opportunities I saw here, eventually led me to the topic of this Master thesis.

The process of implementing and writing this thesis itself certainly had its ups and downs. A new challenge was to plan and implement such a long project all by myself. At times, this gave me a lot of freedom, without any pressing deadlines every two weeks. However, it also came at the expense of long working hours at other times. Especially challenging were problems in the implementation that I did not foresee. An example for this is the data conversion in FME which was extremely time consuming with a lot of bugs and errors in between. Nevertheless, I am also proud of how fast my knowledge in such a complex topic grew. Without forgetting about the bigger picture, I was also able to think about small details and implications coming with them.

Altogether, the thesis asked for a lot of stamina and creativity. Now looking back at what I achieved in this time, I can confidently say that I am happy with this experience and with what I've learnt through this.

Literature

Abbasabadi, N., & Ashayeri, M. (2019). Urban energy use modeling methods and tools: A review and an outlook. *Building and Environment*, *161*, 106270. https://doi.org/10.1016/j.buildenv.2019.106270

Agugiaro, 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*(1), 2. https://doi.org/10.1186/s40965-018-0042-y

Akahoshi, K., Ishimaru, N., Kurokawa, C., Tanaka, Y., Oishi, T., Kutzner, T., & Kolbe, T. H. (2020). i-Urban revitalization: Conceptual modeling, implementation, and visualization towards sustainable urban planning using CityGML. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *V-4–2020*, 179–186. https://doi.org/10.5194/isprs-annals-V-4-2020-179-2020

Arroyo Ohori, K., Ledoux, H., & Peters, R. (2022). *3D modelling of the built environment: Vol. v0.8*. https://github.com/tudelft3d/3dbook/releases

Becker, T., Nagel, C., & Kolbe, T. H. (2011). Integrated 3D Modeling of Multi-utility Networks and Their Interdependencies for Critical Infrastructure Analysis. In T. H. Kolbe, G. König, & C. Nagel (Eds.), *Advances in 3D Geo-Information Sciences* (pp. 1–20). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-12670-3_1

Beil, C., Kutzner, T., Schwab, B., Willenborg, B., Gawronski, A., & Kolbe, T. H. (2021). Integration of 3D Point Clouds with Semantic 3D City Models—Providing Semantic Information beyond Classification. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *VIII-4/W2-2021*, 105–112. https://doi.org/10.5194/isprs-annals-VIII-4-W2-2021-105-2021

Beil, C., Ruhdorfer, R., Coduro, T., & Kolbe, T. H. (2020). Detailed Streetspace Modelling for Multiple Applications: Discussions on the Proposed CityGML 3.0 Transportation

Model. *ISPRS International Journal of Geo-Information*, *9*(10), 603. https://doi.org/10.3390/ijgi9100603

Biljecki, F., Kumar, K., & Nagel, C. (2018). CityGML Application Domain Extension (ADE): Overview of developments. *Open Geospatial Data, Software and Standards*, *3*(1), 13. https://doi.org/10.1186/s40965-018-0055-6

Biljecki, F., Ledoux, H., & Stoter, J. (2016). An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, *59*, 25–37. https://doi.org/10.1016/j.compenvurbsys.2016.04.005

Biljecki, F., Lim, J., Crawford, J., Moraru, D., Tauscher, H., Konde, A., Adouane, K., Lawrence, S., Janssen, P., & Stouffs, R. (2021). Extending CityGML for IFC-sourced 3D city models. *Automation in Construction*, *121*, 103440. https://doi.org/10.1016/j.autcon.2020.103440

Breu, R., Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., & Thurner, V. (1997). Towards a formalization of the Unified Modeling Language. In M. Akşit & S. Matsuoka (Eds.), *ECOOP'97—Object-Oriented Programming* (Vol. 1241, pp. 344–366). Springer Berlin Heidelberg. https://doi.org/10.1007/BFb0053386

Chaturvedi, K., & Kolbe, T. H. (2017). *Future City Pilot 1 Engineering Report*. http://docs.opengeospatial.org/per/16-098.html#SOSTool

Coors, V., Pietruschka, D., & Zeitler, B. (Eds.). (2022). *iCity. Transformative Research for the Livable, Intelligent, and Sustainable City: Research Findings of University of Applied Sciences Stuttgart*. Springer International Publishing. https://doi.org/10.1007/978-3-030-92096-8

Coors, V., Rodrigues, P., Weiler, V., Duminil, E., Klöber, A., Holweg, D., Brüggemann, T., Bohn, K., Groll, L., Balbach, B., & Spath, F. (2021). *SimStadt 2.0: Schlussbericht*. Hochschule für Technik Stuttgart; https://www.tib.eu/de/suchen/id/TIBKAT%3A1773515144

Corrado, V., & Fabrizio, E. (2019). Steady-State and Dynamic Codes, Critical Review, Advantages and Disadvantages, Accuracy, and Reliability. In *Handbook of Energy Efficiency in Buildings* (pp. 263–294). Elsevier. https://doi.org/10.1016/B978-0-12-812817-6.00011-5

Dalla Costa, S., Roccatello, E., & Rumor, M. (2011). A CityGML 3D geodatabase for buildings' energy efficiency. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XXXVIII-4/C21*, 19–24. https://doi.org/10.5194/isprsarchives-XXXVIII-4-C21-19-2011

European Commission. (2020). *Energy efficiency in buildings*. European Commission - Energy Department. https://ec.europa.eu/info/sites/default/files/energy_climate_change_environment/events/documents/in_focus_energy_efficiency_in_buildings_en.pdf

Geiger, A., Nichersu, A., & Hagenmeyer, V. (2020). Sensitivity of input data in building heating energy demand simulation. *BauSIM 2020, 23. - 25. September 2020, Graz*. BauSIM.

Gröger, G., Kolbe, T. H., Nagel, C., & Häfele, K. H. (2012). *OGC city geography markup language (CityGML) encoding standard. Reference number: OGC 12-019*. http://www.opengis.net/spec/citygml/2.0

Gröger, G., & Plümer, L. (2012). CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, *71*, 12–33. https://doi.org/10.1016/j.isprsjprs.2012.04.004

Horak, D., Hainoun, A., Neugebauer, G., & Stoeglehner, G. (2022). A review of spatio-temporal urban energy system modeling for urban decarbonization strategy formulation. *Renewable and Sustainable Energy Reviews*, *162*, 112426. https://doi.org/10.1016/j.rser.2022.112426

INSPIRE Thematic Working Group Buildings. (2013). *D2.8.III.2 INSPIRE Data Specification on Buildings – Technical Guidelines*. European Commission Joint Research Centre. https://inspire.ec.europa.eu/documents/Data_Specifications/INSPIRE_DataSpecification_BU_v3.0.pdf

Kämpf, J., & Coccolo, S. (2015). *Concrete application case of CitySim: Presentation for the CityGML Energy ADE Workshop in Munich*. https://en.wiki.energy.sig3d.org/images/upload/EnergyADE_CitySim_30.11.2015.pdf

Kolbe, T. H., Kutzner, T., Smyth, C. S., Nagel, C., Roensdorf, C., & Heazel, C. (2021). *OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard. Reference number: 20-010*. http://www.opengis.net/doc/IS/CityGML-1/3.0

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, *88*(1), 43–61. https://doi.org/10.1007/s41064-020-00095-z

Kutzner, T., Hijazi, I., & Kolbe, T. H. (2018). Semantic Modelling of 3D Multi-Utility Networks for Urban Analyses and Simulations: The CityGML Utility Network ADE. *International Journal of 3-D Information Modeling*, *7*(2), 1–34. https://doi.org/10.4018/IJ3DIM.2018040101

Kutzner, T., & Kolbe, T. H. (2018). CityGML 3.0: Sneak preview. *PFGK18-Photogrammetrie-Fernerkundung-Geoinformatik-Kartographie, 37. Jahrestagung in München 2018*, 835–839.

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., & Vitalis, S. (2019). CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, *4*(1), 4. https://doi.org/10.1186/s40965-019-0064-0

León-Sánchez, C., Giannelli, D., Agugiaro, G., & Stoter, J. (2021). Testing the new 3D BAG Dataset for Energy Demand Estimation of Residential Buildings. *The International*

Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, *XLVI-4/W1-2021*, 69–76. https://doi.org/10.5194/isprs-archives-XLVI-4-W1-2021-69-2021

Löwner, M.-O., Benner, J., & Gröger, G. (2014). Aktuelle trends in der Entwicklung von CityGML 3.0. *Geoinformationen Öffnen Das Tor Zur Welt*, *34*.

Löwner, M.-O., & Gröger, G. (2017). Das neue LoD Konzept für CityGML 3.0. *Proc. 13th GeoForum MV. GeoMV EV, Warnemünde, Germany*, 1–8.

Malhotra, A., Shamovich, M., Frisch, J., & van Treeck, C. (2019). *Parametric Study of the Different Level of Detail of CityGML and Energy-ADE Information for Energy Performance Simulations*. 3429–3436. https://doi.org/10.26868/25222708.2019.210607

Nageler, P., Koch, A., Mauthner, F., Leusbrock, I., Mach, T., Hochenauer, C., & Heimrath, R. (2018). Comparison of dynamic urban building energy models (UBEM): Sigmoid energy signature and physical modelling approach. *Energy and Buildings*, *179*, 333–343. https://doi.org/10.1016/j.enbuild.2018.09.034

Nega, A., & Coors, V. (2022). The Use of CityGML 3.0 in 3D Cadastre system: The Case of Addis Ababa City. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVIII-4/W4-2022*, 109–116. https://doi.org/10.5194/isprs-archives-XLVIII-4-W4-2022-109-2022

Object Management Group Unified Modeling Language (OMG UML). (2012). ISO. https://www.iso.org/standard/52854.html

Pasquinelli, A., Agugiaro, G., Tagliabue, L., Scaioni, M., & Guzzetti, F. (2019). Exploiting the Potential of Integrated Public Building Data: Energy Performance Assessment of the Building Stock in a Case Study in Northern Italy. *ISPRS International Journal of Geo-Information*, *8*(1), 27. https://doi.org/10.3390/ijgi8010027

Pilone, D., & Pitman, N. (2005). *UML 2.0 in a nutshell* (1st ed). O'Reilly Media.

Prieto, I., Izkara, J. L., & Delgado del Hoyo, F. J. (2012). Efficient Visualization of the Geometric Information of CityGML: Application for the Documentation of Built Heritage. In B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, & B. O. Apduhan (Eds.), *Computational Science and Its Applications – ICCSA 2012* (Vol. 7333, pp. 529–544). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31125-3_40

Reinhart, C. F., & Cerezo Davila, C. (2016). Urban building energy modeling – A review of a nascent field. *Building and Environment*, *97*, 196–202. https://doi.org/10.1016/j.buildenv.2015.12.001

Ribeiro, H. V., Rybski, D., & Kropp, J. P. (2019). Effects of changing population or density on urban carbon dioxide emissions. *Nature Communications*, *10*(1), 3204. https://doi.org/10.1038/s41467-019-11184-y

Rossknecht, M., & Airaksinen, E. (2020). Concept and Evaluation of Heating Demand Prediction Based on 3D City Models and the CityGML Energy ADE—Case Study Helsinki. *ISPRS International Journal of Geo-Information*, *9*(10), 602. https://doi.org/10.3390/ijgi9100602

Rules for application schema. (2015). ISO. https://www.iso.org/standard/59193.html

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The unified modeling language reference manual*. Addison-Wesley. http://debracollege.dspaces.org/bitstream/123456789/404/1/UML%20Reference%20Manual%20by%20James%20Rambaugh.pdf

Saeidian, B., Rajabifard, A., Atazadeh, B., & Kalantari, M. (2022). Extending CityGML 3.0 to Support 3D Underground Land Administration. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVIII-4/W4-2022*, 125–132. https://doi.org/10.5194/isprs-archives-XLVIII-4-W4-2022-125-2022

Schildt, M., Behm, C., Malhotra, A., Weck-Ponten, S., Frisch, J., & Treeck, C. (2021, September). *Proposed Integration of Utilities in the Energy ADE 2.0*.

Tufan, Ö. (2022). *Development and Testing of the CityJSON Energy Extension for Space Heating Demand Calculation* [Master Thesis]. Delft University of Technology.

Unified Modeling Language Version 2.5.1. (2017). OMG. https://www.omg.org/spec/UML

van den Brink, L., Stoter, J., & Zlatanova, S. (2013). UML-Based Approach to Developing a CityGML Application Domain Extension: UML-Based Approach to Developing a CityGML Application Domain Extension. *Transactions in GIS*, *17*(6), 920–942. https://doi.org/10.1111/tgis.12026

Yan, J., Zlatanova, S., & Diakité, A. (2021). A unified 3D space-based navigation model for seamless navigation in indoor and outdoor. *International Journal of Digital Earth*, *14*(8), 985–1003. https://doi.org/10.1080/17538947.2021.1913522

Online References

3DCityDB Energy ADE [Git Repository]. Retrieved from https://github.com/3dcitydb/energy-ade-citydb. Last accessed: 13.11.2022.

CityGML-3.0Encodings [Git Repository]. Retrieved from https://github.com/opengeospatial/CityGML-3.0Encodings/. Last accessed: 10.11.2022.

CityGML3.0-GML-Encoding [Git Repository]. Retrieved from https://github.com/opengeospatial/CityGML3.0-GML-Encoding. Last accessed: 12.11.2022.

citygml3-utility-network-ade [Git Repository]. Retrieved from https://github.com/tum-gis/citygml3-utility-network-ade. Last accessed: 26.11.2022.

Citygml4j [Git Repository]. Retrieved from https://github.com/citygml4j/citygml4j. Last accessed: 12.11.2022.

CityJSON [Website]. Retrieved from https://www.cityjson.org/. Last accessed: 10.11.2022.

CityJSON Specification [Specification]. Retrieved from https://www.cityjson.org/specs/1.1.1/. Last accessed: 10.11.2022.

Energy ADE [Git Repository]. Retrieved from https://git.rwth-aachen.de/energyade/citygml-energy. Last accessed: 07.11.2022.

Enterprise Architect [Website]. Retrieved from https://sparxsystems.com/. Last accessed: 15.11.2022.

FME conversion [Website]. Retrieved from https://hub.safe.com/publishers/con-terra/templates/convert-citygml-2-0-to-3-0. Last accessed: 12.11.2022.

gbXML [Website]. Retrieved from: https://www.gbxml.org/. Last accessed: 02.01.2023.

KIT Profile [Website]. Retrieved from https://www.citygmlwiki.org/upload/EnergyADE%201.0/KIT-Profile/FeatureCatalogue/. Last accessed: 13.11.2022.

OGC CityGML [Website]. Retrieved from https://www.ogc.org/standards/citygml. Last accessed: 10.11.2022.

ShapeChange [Website]. Retrieved from https://shapechange.net/. Last accessed: 02.01.2022.

Sustainable Development Goals [Website]. Retrieved from https://www.un.org/sustainabledevelopment/cities/. Last accessed: 29.11.2022.

TestADE [Git Repository]. Retrieved from https://github.com/3dcitydb/extension-test-ade. Last accessed: 12.10.2022.

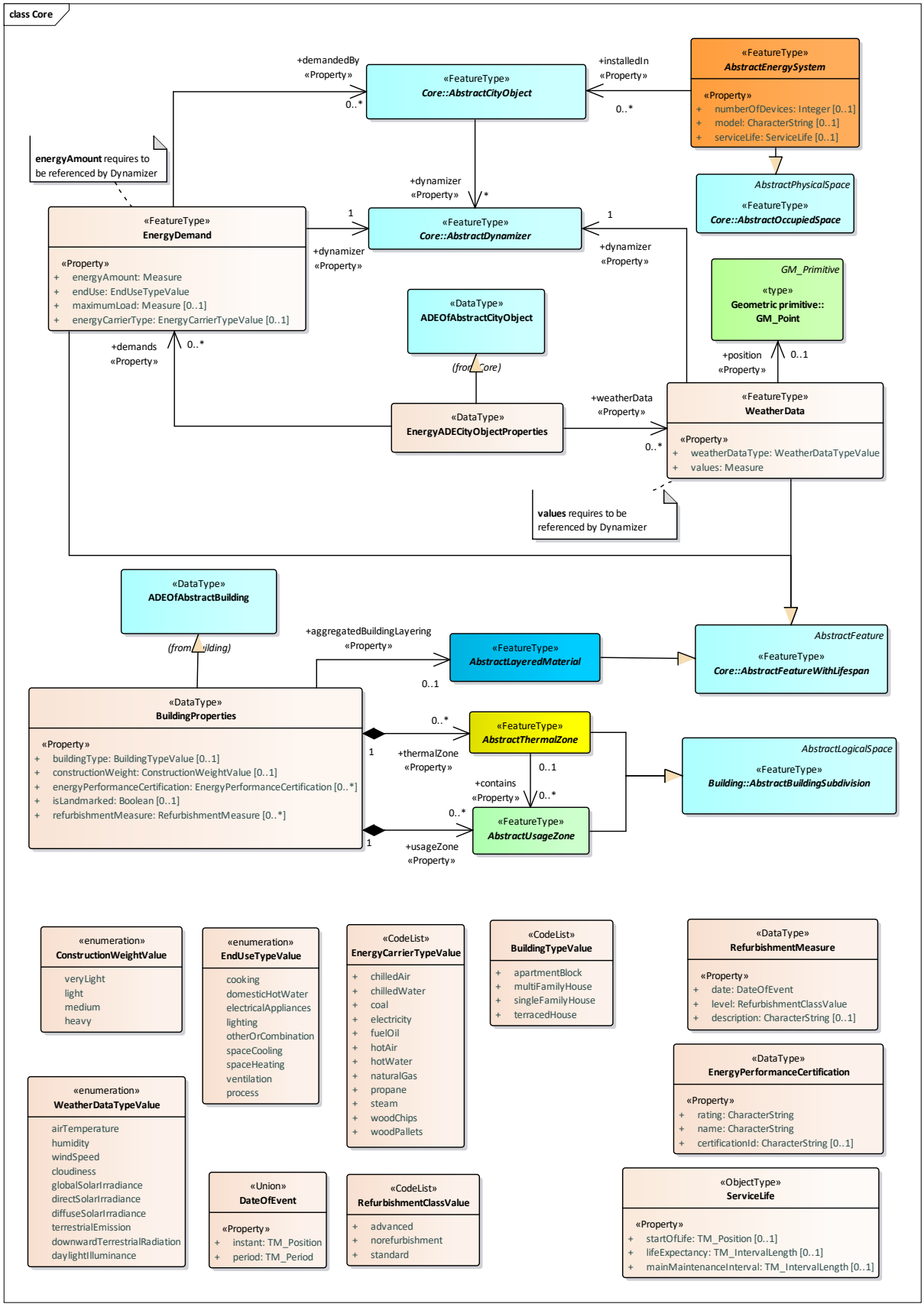
Urban Planning ADE [Git Repository]. Retrieved from https://github.com/opengeospatial/CityGML3.0-GML-Encoding/tree/main/resources/examples/ADE-examples/Urban-Planning-ADE. Last accessed: 12.11.2022.

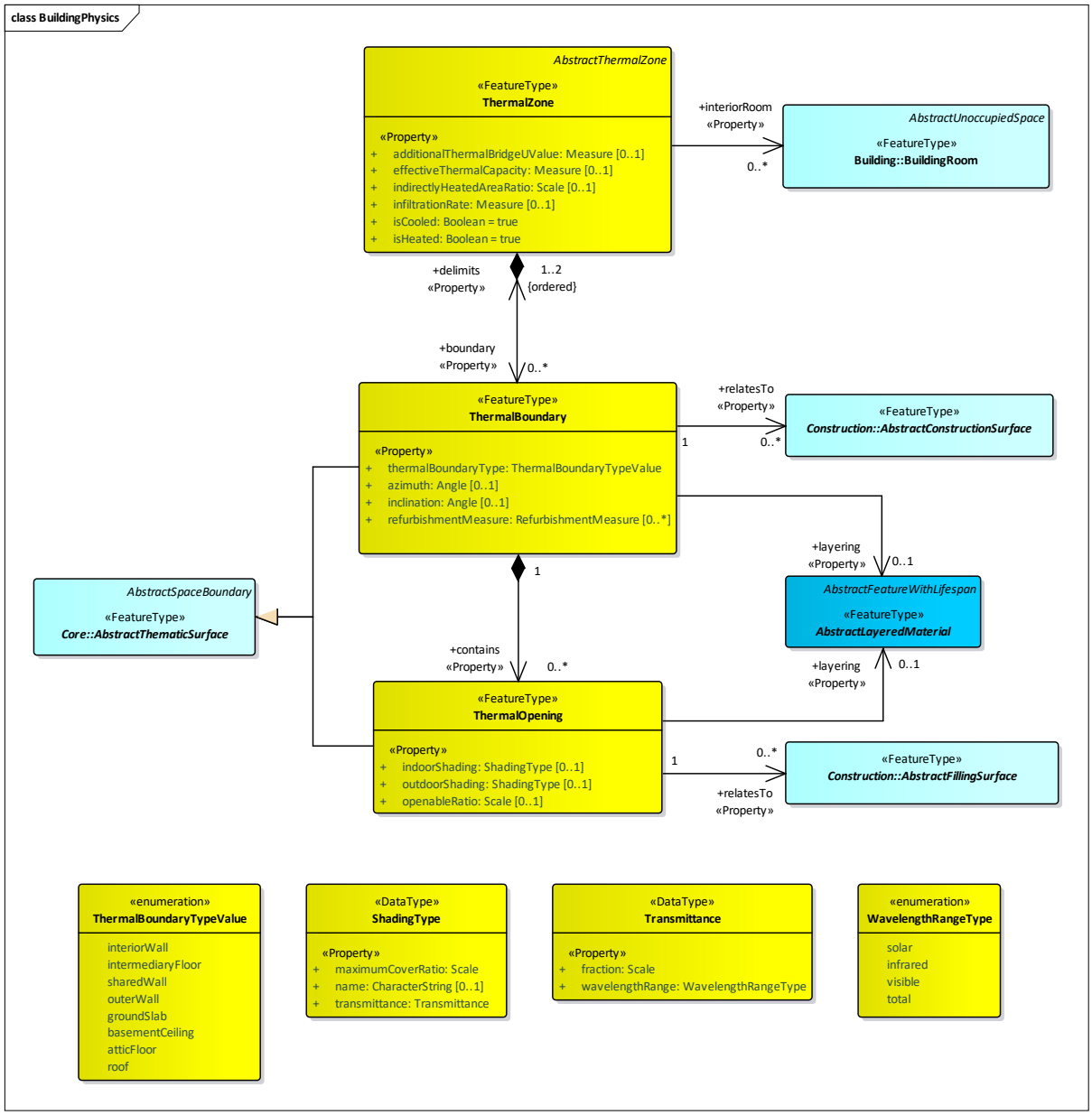
UtilityNetworkADE [Git Repository]. Retrieved from https://github.com/TatjanaKutzner/CityGML-UtilityNetwork-ADE. Last accessed: 12.10.2022.

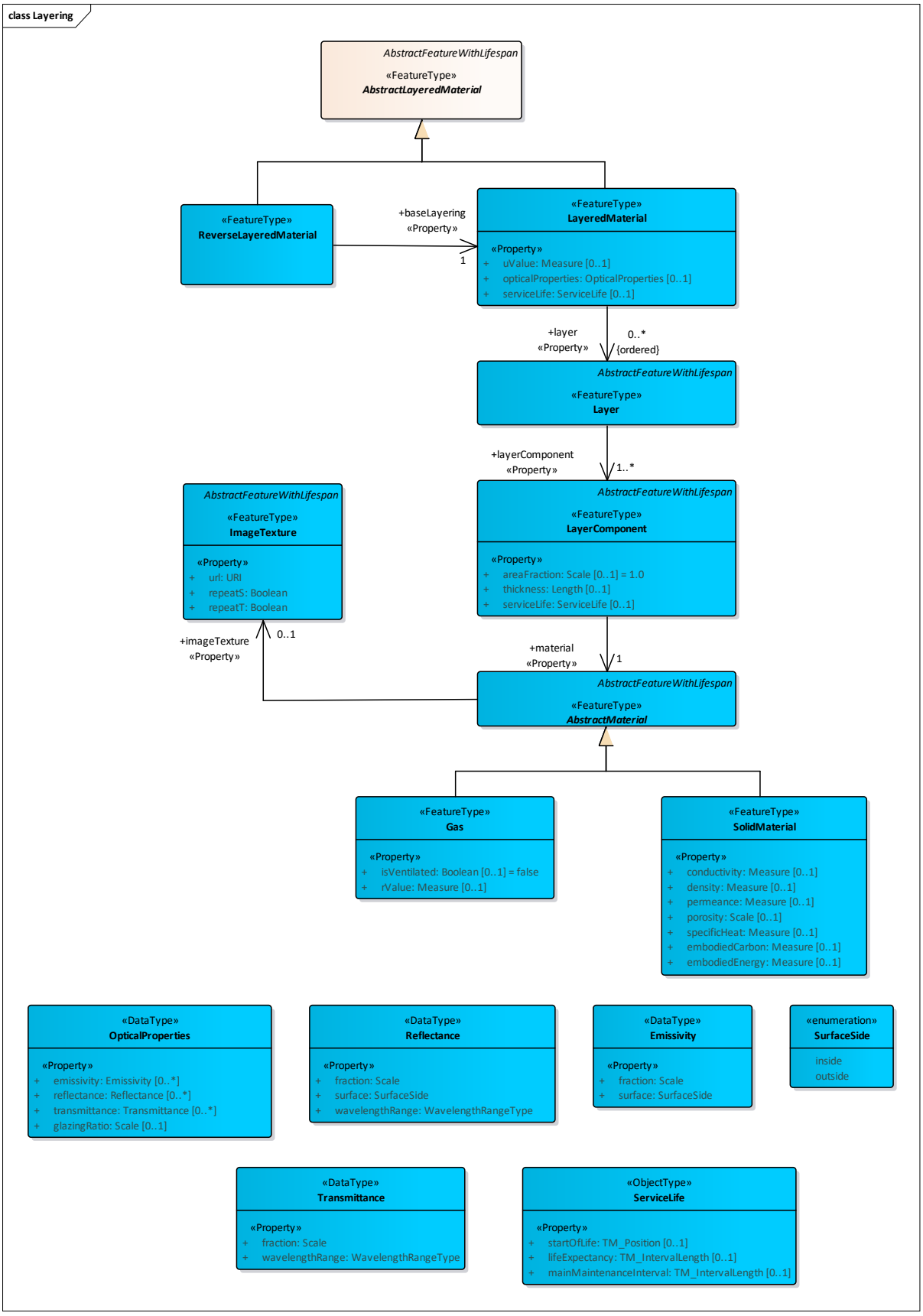
World Bank (2020a). Rural Population [Data file]. Retrieved from https://data.worldbank.org/indicator/SP.RUR.TOTL. Last accessed: 29.11.2022.

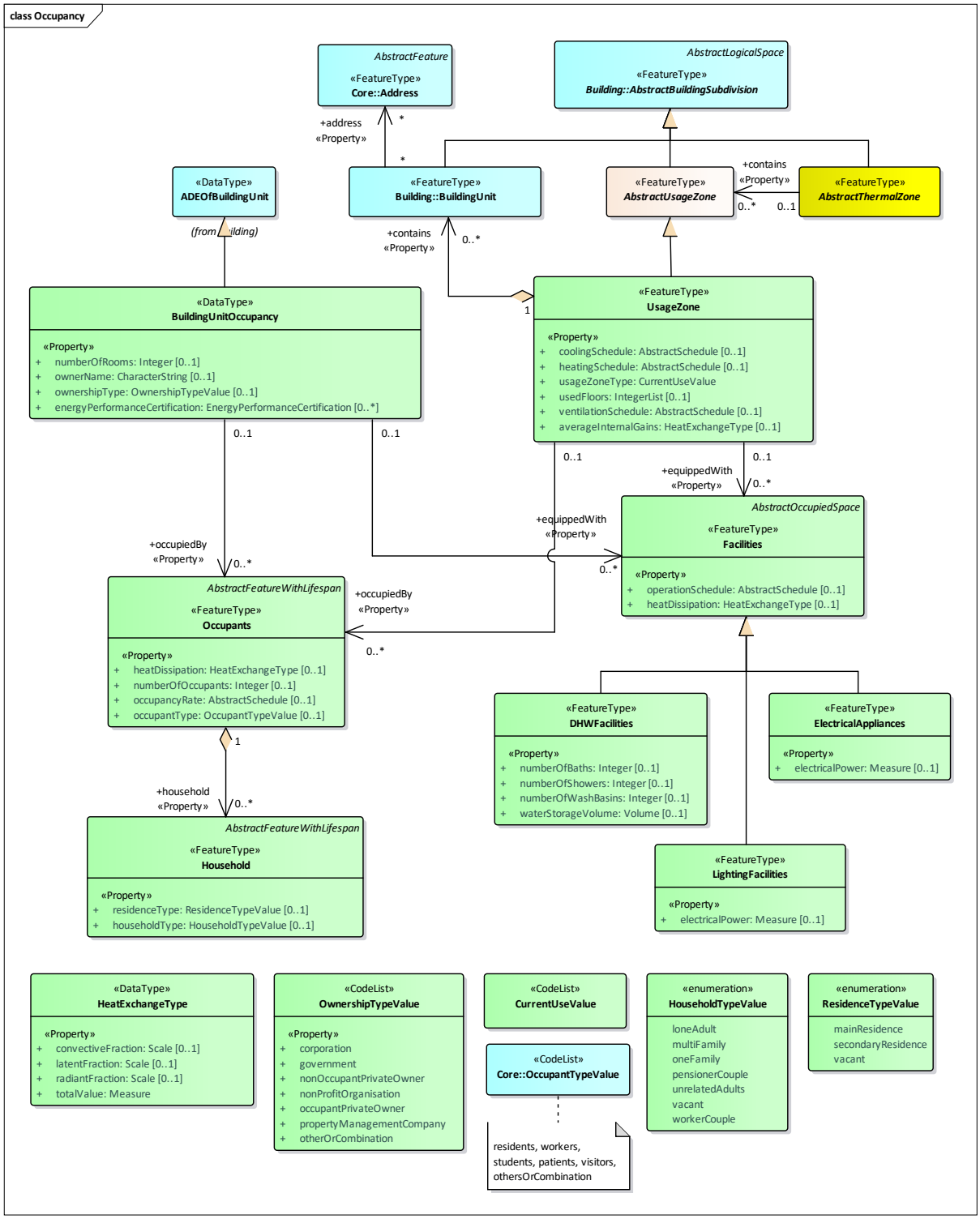
World Bank (2020b). Urban Population [Data file]. Retrieved from https://data.worldbank.org/indicator/SP.URB.TOTL. Last accessed: 29.11.2022.

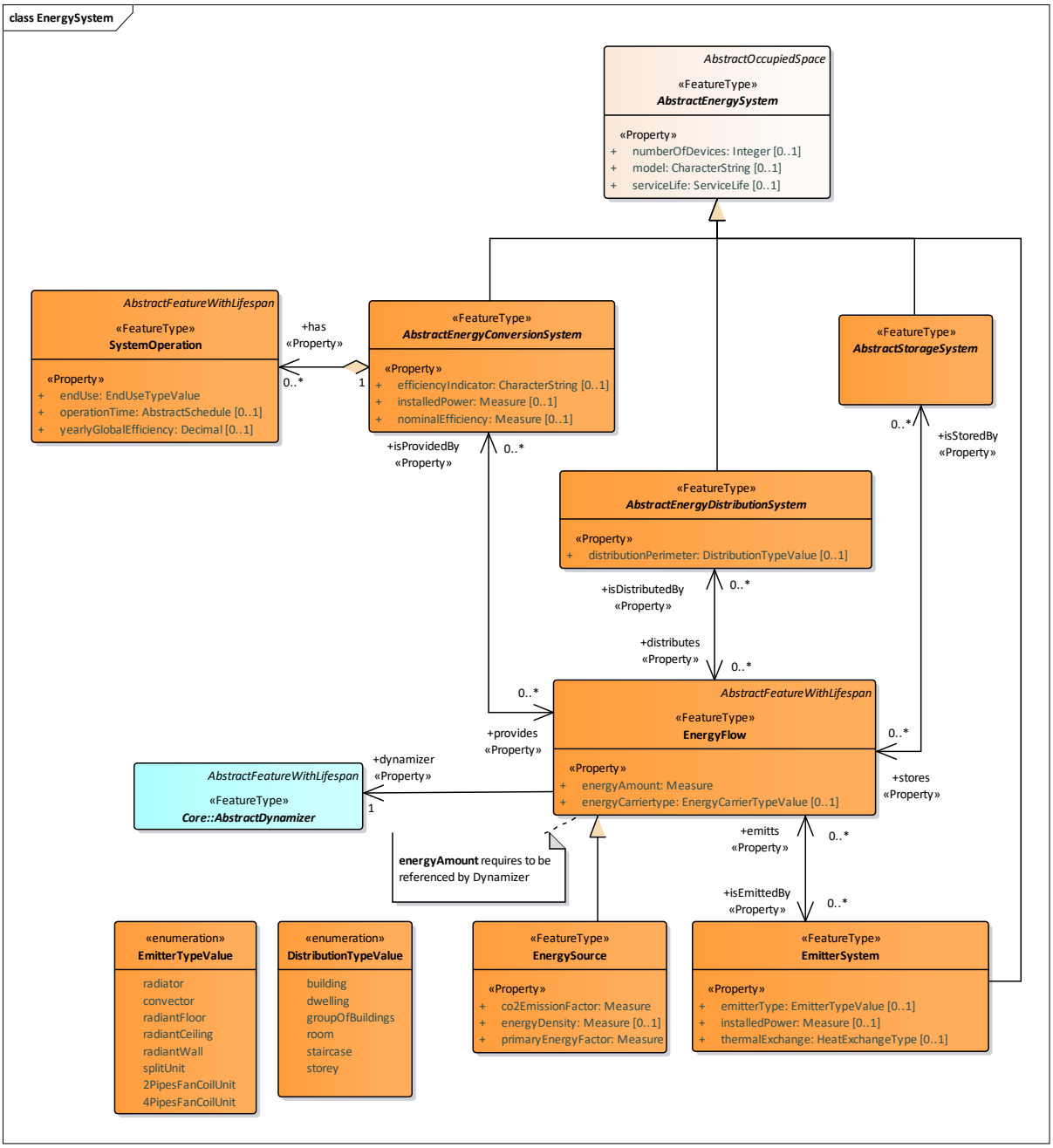
Appendix A: UML diagrams of Energy ADE for CityGML 3.0

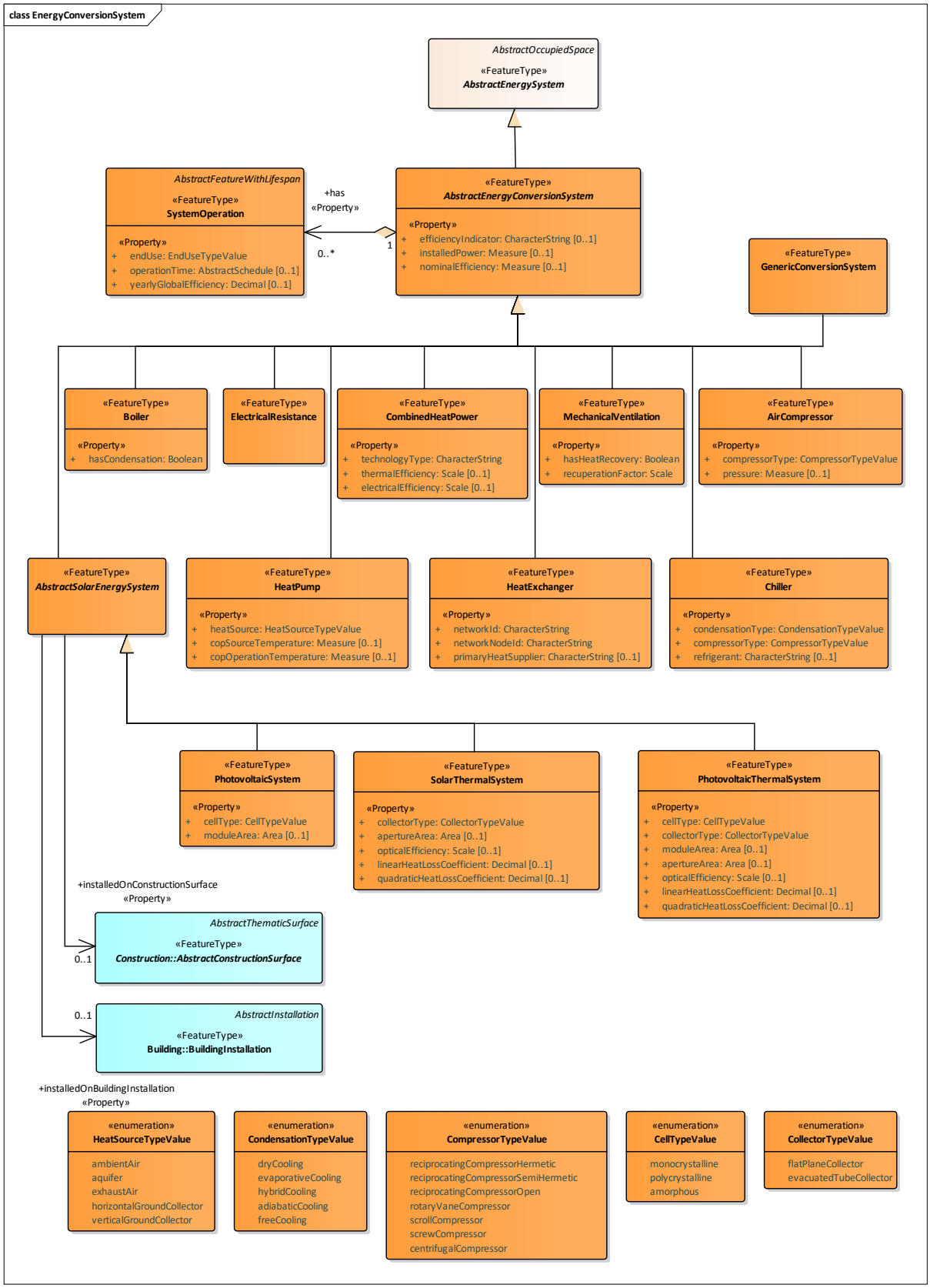


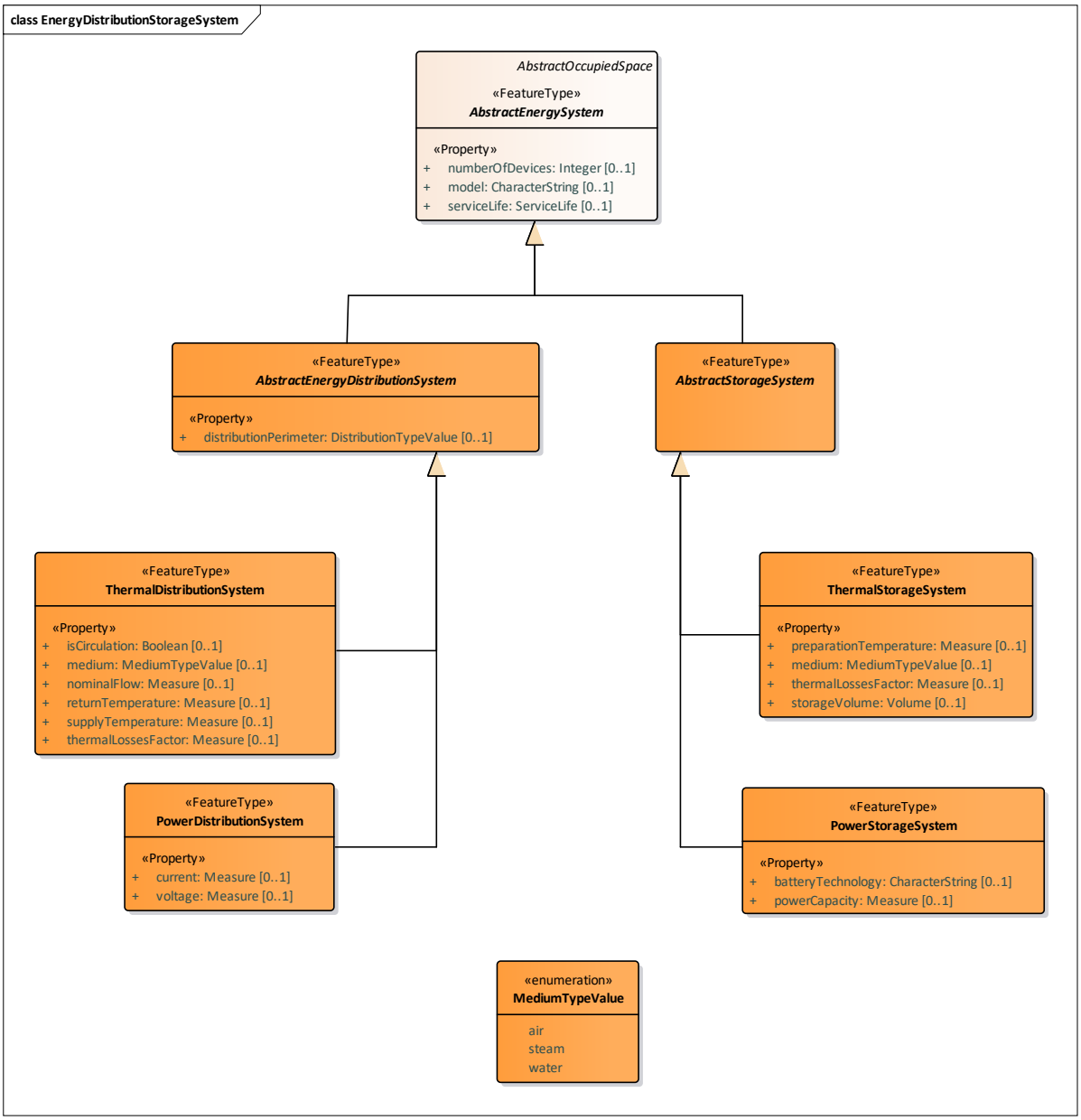

class BuildingPhysics
AbstractThermalZone
«FeatureType»
ThermalZone
«Property»
+ additionalThermalBridgeUValue: Measure [0..1]
+ effectiveThermalCapacity: Measure [0..1]
+ indirectlyHeatedAreaRatio: Scale [0..1]
+ infiltrationRate: Measure [0..1]
+ isCooled: Boolean = true
+ isHeated: Boolean = true
+interiorRoom
«Property»
0..*
AbstractUnoccupiedSpace
«FeatureType»
Building::BuildingRoom
+delimits
«Property»
1..2
{ordered}
+boundary
«Property»
0..*
«FeatureType»
ThermalBoundary
«Property»
+ thermalBoundaryType: ThermalBoundaryTypeValue
+ azimuth: Angle [0..1]
+ inclination: Angle [0..1]
+ refurbishmentMeasure: RefurbishmentMeasure [0..*]
+relatesTo
«Property»
1
0..*
«FeatureType»
Construction::AbstractConstructionSurface
AbstractSpaceBoundary
«FeatureType»
Core::AbstractThematicSurface
1
+contains
«Property»
0..*
+layering
«Property»
0..1
AbstractFeatureWithLifespan
«FeatureType»
AbstractLayeredMaterial
+layering
«Property»
0..1
«FeatureType»
ThermalOpening
«Property»
+ indoorShading: ShadingType [0..1]
+ outdoorShading: ShadingType [0..1]
+ openableRatio: Scale [0..1]
1
0..*
+relatesTo
«Property»
«FeatureType»
Construction::AbstractFillingSurface
«enumeration»
ThermalBoundaryTypeValue
interiorWall
intermediaryFloor
sharedWall
outerWall
groundSlab
basementCeiling
atticFloor
roof
«DataType»
ShadingType
«Property»
+ maximumCoverRatio: Scale
+ name: CharacterString [0..1]
+ transmittance: Transmittance
«DataType»
Transmittance
«Property»
+ fraction: Scale
+ wavelengthRange: WavelengthRangeType
«enumeration»
WavelengthRangeType
solar
infrared
visible
total

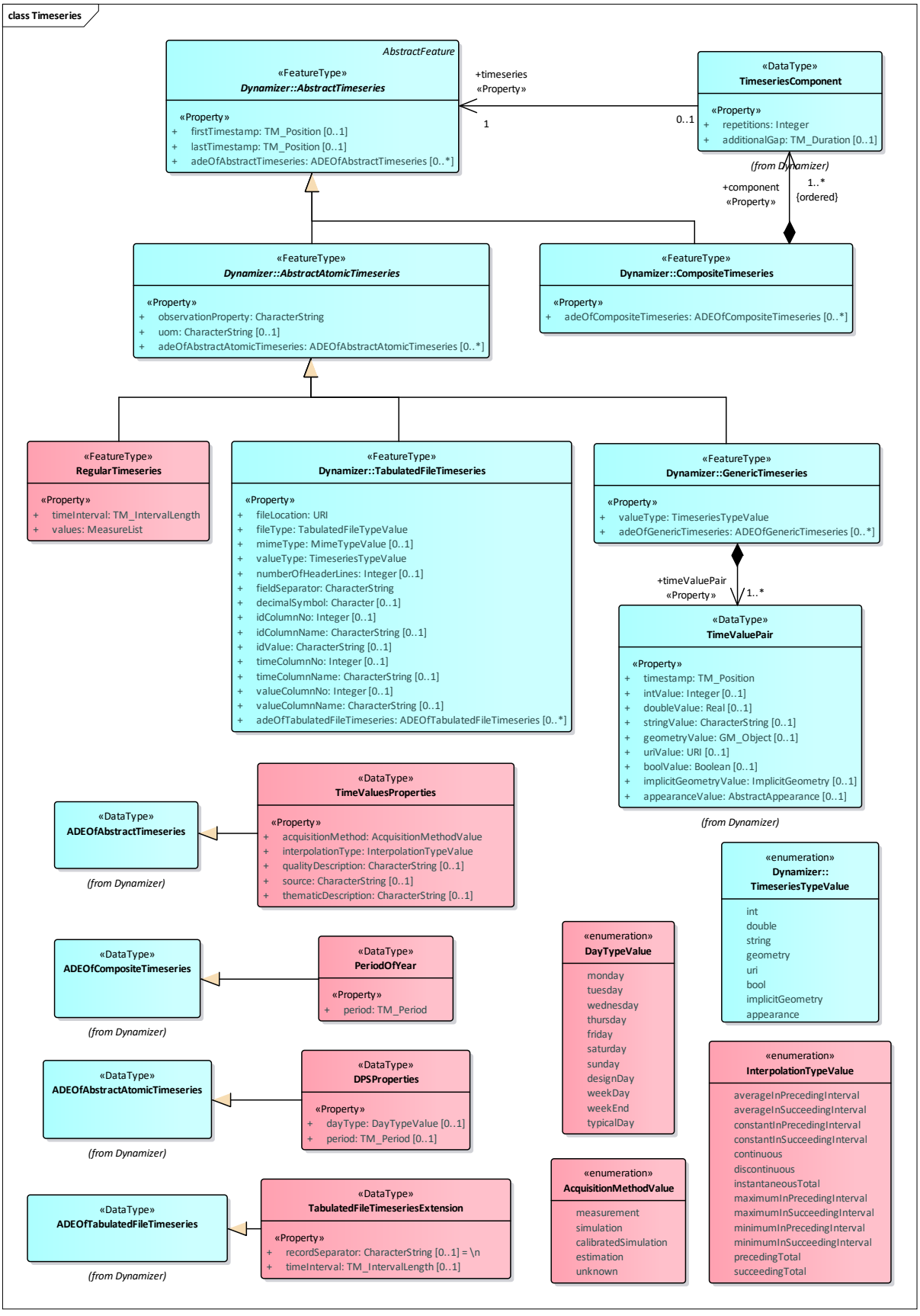

class Layering
AbstractFeatureWithLifespan
«FeatureType»
AbstractLayeredMaterial
«FeatureType»
ReverseLayeredMaterial
+baseLayering
«Property»
1
«FeatureType»
LayeredMaterial
«Property»
+ uValue: Measure [0..1]
+ opticalProperties: OpticalProperties [0..1]
+ serviceLife: ServiceLife [0..1]
+layer
«Property»
0..*
{ordered}
AbstractFeatureWithLifespan
«FeatureType»
Layer
+layerComponent
«Property»
1..*
AbstractFeatureWithLifespan
«FeatureType»
ImageTexture
«Property»
+ url: URI
+ repeatS: Boolean
+ repeatT: Boolean
AbstractFeatureWithLifespan
«FeatureType»
LayerComponent
«Property»
+ areaFraction: Scale [0..1] = 1.0
+ thickness: Length [0..1]
+ serviceLife: ServiceLife [0..1]
+imageTexture
«Property»
0..1
+material
«Property»
1
AbstractFeatureWithLifespan
«FeatureType»
AbstractMaterial
«FeatureType»
Gas
«Property»
+ isVentilated: Boolean [0..1] = false
+ rValue: Measure [0..1]
«FeatureType»
SolidMaterial
«Property»
+ conductivity: Measure [0..1]
+ density: Measure [0..1]
+ permeance: Measure [0..1]
+ porosity: Scale [0..1]
+ specificHeat: Measure [0..1]
+ embodiedCarbon: Measure [0..1]
+ embodiedEnergy: Measure [0..1]
«DataType»
OpticalProperties
«Property»
+ emissivity: Emissivity [0..*]
+ reflectance: Reflectance [0..*]
+ transmittance: Transmittance [0..*]
+ glazingRatio: Scale [0..1]
«DataType»
Reflectance
«Property»
+ fraction: Scale
+ surface: SurfaceSide
+ wavelengthRange: WavelengthRangeType
«DataType»
Emissivity
«Property»
+ fraction: Scale
+ surface: SurfaceSide
«enumeration»
SurfaceSide
inside
outside
«DataType»
Transmittance
«Property»
+ fraction: Scale
+ wavelengthRange: WavelengthRangeType
«ObjectType»
ServiceLife
«Property»
+ startOfLife: TM_Position [0..1]
+ lifeExpectancy: TM_IntervalLength [0..1]
+ mainMaintenanceInterval: TM_IntervalLength [0..1]

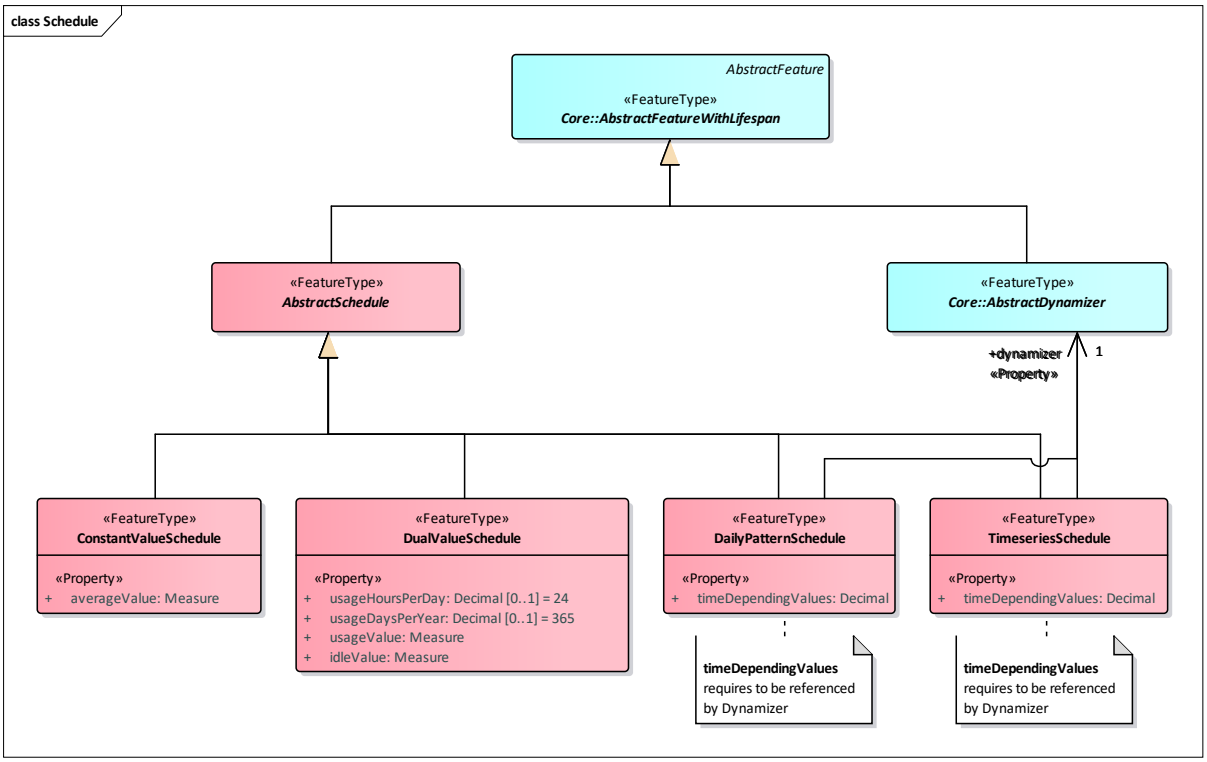

class Occupancy
AbstractFeature
«FeatureType»
Core::Address
AbstractLogicalSpace
«FeatureType»
Building::AbstractBuildingSubdivision
+address
«Property»
*
*
«DataType»
ADEOfBuildingUnit
(from Building)
«FeatureType»
Building::BuildingUnit
«FeatureType»
AbstractUsageZone
+contains
«Property»
0..*
0..1
«FeatureType»
AbstractThermalZone
+contains
«Property»
0..*
«DataType»
BuildingUnitOccupancy
«Property»
+ numberOfRooms: Integer [0..1]
+ ownerName: CharacterString [0..1]
+ ownershipType: OwnershipTypeValue [0..1]
+ energyPerformanceCertification: EnergyPerformanceCertification [0..*]
1
«FeatureType»
UsageZone
«Property»
+ coolingSchedule: AbstractSchedule [0..1]
+ heatingSchedule: AbstractSchedule [0..1]
+ usageZoneType: CurrentUseValue
+ usedFloors: IntegerList [0..1]
+ ventilationSchedule: AbstractSchedule [0..1]
+ averageInternalGains: HeatExchangeType [0..1]
0..1
0..1
0..1
0..1
+equippedWith
«Property»
0..*
+occupiedBy
«Property»
0..*
+equippedWith
«Property»
0..*
AbstractOccupiedSpace
«FeatureType»
Facilities
«Property»
+ operationSchedule: AbstractSchedule [0..1]
+ heatDissipation: HeatExchangeType [0..1]
AbstractFeatureWithLifespan
«FeatureType»
Occupants
«Property»
+ heatDissipation: HeatExchangeType [0..1]
+ numberOfOccupants: Integer [0..1]
+ occupancyRate: AbstractSchedule [0..1]
+ occupantType: OccupantTypeValue [0..1]
+occupiedBy
«Property»
0..*
1
+household
«Property»
0..*
«FeatureType»
DHWFacilities
«Property»
+ numberOfBaths: Integer [0..1]
+ numberOfShowers: Integer [0..1]
+ numberOfWashBasins: Integer [0..1]
+ waterStorageVolume: Volume [0..1]
«FeatureType»
ElectricalAppliances
«Property»
+ electricalPower: Measure [0..1]
AbstractFeatureWithLifespan
«FeatureType»
Household
«Property»
+ residenceType: ResidenceTypeValue [0..1]
+ householdType: HouseholdTypeValue [0..1]
«FeatureType»
LightingFacilities
«Property»
+ electricalPower: Measure [0..1]
«DataType»
HeatExchangeType
«Property»
+ convectiveFraction: Scale [0..1]
+ latentFraction: Scale [0..1]
+ radiantFraction: Scale [0..1]
+ totalValue: Measure
«CodeList»
OwnershipTypeValue
«Property»
+ corporation
+ government
+ nonOccupantPrivateOwner
+ nonProfitOrganisation
+ occupantPrivateOwner
+ propertyManagementCompany
+ otherOrCombination
«CodeList»
CurrentUseValue
«CodeList»
Core::OccupantTypeValue
residents, workers,
students, patients, visitors,
othersOrCombination
«enumeration»
HouseholdTypeValue
loneAdult
multiFamily
oneFamily
pensionerCouple
unrelatedAdults
vacant
workerCouple
«enumeration»
ResidenceTypeValue
mainResidence
secondaryResidence
vacant

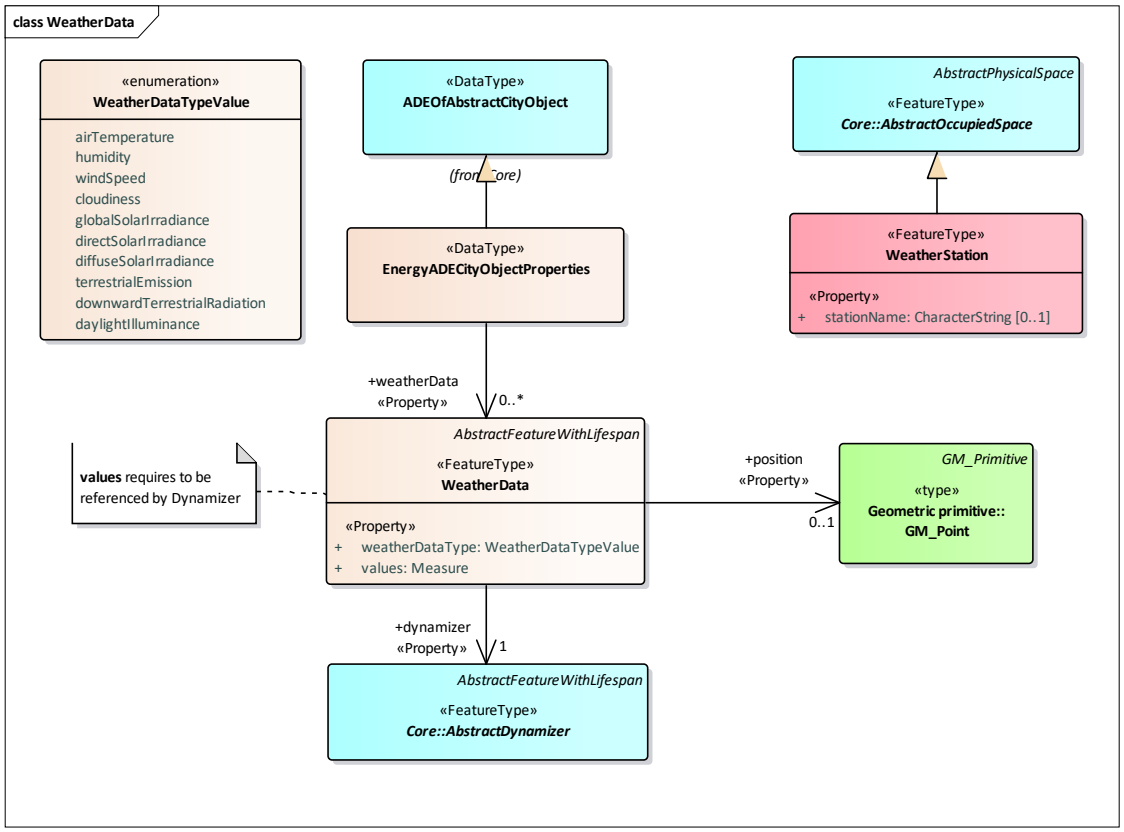

class EnergySystem
AbstractOccupiedSpace
«FeatureType»
AbstractEnergySystem
«Property»
+ numberOfDevices: Integer [0..1]
+ model: CharacterString [0..1]
+ serviceLife: ServiceLife [0..1]
AbstractFeatureWithLifespan
«FeatureType»
SystemOperation
«Property»
+ endUse: EndUseTypeValue
+ operationTime: AbstractSchedule [0..1]
+ yearlyGlobalEfficiency: Decimal [0..1]
+has
«Property»
0..*
1
«FeatureType»
AbstractEnergyConversionSystem
«Property»
+ efficiencyIndicator: CharacterString [0..1]
+ installedPower: Measure [0..1]
+ nominalEfficiency: Measure [0..1]
«FeatureType»
AbstractStorageSystem
0..*
+isStoredBy
«Property»
+isProvidedBy
«Property»
0..*
«FeatureType»
AbstractEnergyDistributionSystem
«Property»
+ distributionPerimeter: DistributionTypeValue [0..1]
+isDistributedBy
«Property»
0..*
+distributes
«Property»
0..*
0..*
+provides
«Property»
AbstractFeatureWithLifespan
«FeatureType»
EnergyFlow
«Property»
+ energyAmount: Measure
+ energyCarriertype: EnergyCarrierTypeValue [0..1]
0..*
+stores
«Property»
AbstractFeatureWithLifespan
«FeatureType»
Core::AbstractDynamizer
+dynamizer
«Property»
1
energyAmount requires to be referenced by Dynamizer
+emitts
«Property»
0..*
+isEmittedBy
«Property»
0..*
«enumeration»
EmitterTypeValue
radiator
convector
radiantFloor
radiantCeiling
radiantWall
splitUnit
2PipesFanCoilUnit
4PipesFanCoilUnit
«enumeration»
DistributionTypeValue
building
dwelling
groupOfBuildings
room
staircase
storey
«FeatureType»
EnergySource
«Property»
+ co2EmissionFactor: Measure
+ energyDensity: Measure [0..1]
+ primaryEnergyFactor: Measure
«FeatureType»
EmitterSystem
«Property»
+ emitterType: EmitterTypeValue [0..1]
+ installedPower: Measure [0..1]
+ thermalExchange: HeatExchangeType [0..1]


class EnergyConversionSystem
AbstractOccupiedSpace
«FeatureType»
AbstractEnergySystem
AbstractFeatureWithLifespan
«FeatureType»
SystemOperation
«Property»
+ endUse: EndUseTypeValue
+ operationTime: AbstractSchedule [0..1]
+ yearlyGlobalEfficiency: Decimal [0..1]
+has
«Property»
0..*
1
«FeatureType»
AbstractEnergyConversionSystem
«Property»
+ efficiencyIndicator: CharacterString [0..1]
+ installedPower: Measure [0..1]
+ nominalEfficiency: Measure [0..1]
«FeatureType»
GenericConversionSystem
«FeatureType»
Boiler
«Property»
+ hasCondensation: Boolean
«FeatureType»
ElectricalResistance
«FeatureType»
CombinedHeatPower
«Property»
+ technologyType: CharacterString
+ thermalEfficiency: Scale [0..1]
+ electricalEfficiency: Scale [0..1]
«FeatureType»
MechanicalVentilation
«Property»
+ hasHeatRecovery: Boolean
+ recuperationFactor: Scale
«FeatureType»
AirCompressor
«Property»
+ compressorType: CompressorTypeValue
+ pressure: Measure [0..1]
«FeatureType»
AbstractSolarEnergySystem
«FeatureType»
HeatPump
«Property»
+ heatSource: HeatSourceTypeValue
+ copSourceTemperature: Measure [0..1]
+ copOperationTemperature: Measure [0..1]
«FeatureType»
HeatExchanger
«Property»
+ networkId: CharacterString
+ networkNodeId: CharacterString
+ primaryHeatSupplier: CharacterString [0..1]
«FeatureType»
Chiller
«Property»
+ condensationType: CondensationTypeValue
+ compressorType: CompressorTypeValue
+ refrigerant: CharacterString [0..1]
«FeatureType»
PhotovoltaicSystem
«Property»
+ cellType: CellTypeValue
+ moduleArea: Area [0..1]
«FeatureType»
SolarThermalSystem
«Property»
+ collectorType: CollectorTypeValue
+ apertureArea: Area [0..1]
+ opticalEfficiency: Scale [0..1]
+ linearHeatLossCoefficient: Decimal [0..1]
+ quadraticHeatLossCoefficient: Decimal [0..1]
«FeatureType»
PhotovoltaicThermalSystem
«Property»
+ cellType: CellTypeValue
+ collectorType: CollectorTypeValue
+ moduleArea: Area [0..1]
+ apertureArea: Area [0..1]
+ opticalEfficiency: Scale [0..1]
+ linearHeatLossCoefficient: Decimal [0..1]
+ quadraticHeatLossCoefficient: Decimal [0..1]
+installedOnConstructionSurface
«Property»
0..1
AbstractThematicSurface
«FeatureType»
Construction::AbstractConstructionSurface
0..1
AbstractInstallation
«FeatureType»
Building::BuildingInstallation
+installedOnBuildingInstallation
«Property»
«enumeration»
HeatSourceTypeValue
ambientAir
aquifer
exhaustAir
horizontalGroundCollector
verticalGroundCollector
«enumeration»
CondensationTypeValue
dryCooling
evaporativeCooling
hybridCooling
adiabaticCooling
freeCooling
«enumeration»
CompressorTypeValue
reciprocatingCompressorHermetic
reciprocatingCompressorSemiHermetic
reciprocatingCompressorOpen
rotaryVaneCompressor
scrollCompressor
screwCompressor
centrifugalCompressor
«enumeration»
CellTypeValue
monocrystalline
polycrystalline
amorphous
«enumeration»
CollectorTypeValue
flatPlaneCollector
evacuatedTubeCollector


class EnergyDistributionStorageSystem
AbstractOccupiedSpace
«FeatureType»
AbstractEnergySystem
«Property»
+ numberOfDevices: Integer [0..1]
+ model: CharacterString [0..1]
+ serviceLife: ServiceLife [0..1]
«FeatureType»
AbstractEnergyDistributionSystem
«Property»
+ distributionPerimeter: DistributionTypeValue [0..1]
«FeatureType»
AbstractStorageSystem
«FeatureType»
ThermalDistributionSystem
«Property»
+ isCirculation: Boolean [0..1]
+ medium: MediumTypeValue [0..1]
+ nominalFlow: Measure [0..1]
+ returnTemperature: Measure [0..1]
+ supplyTemperature: Measure [0..1]
+ thermalLossesFactor: Measure [0..1]
«FeatureType»
ThermalStorageSystem
«Property»
+ preparationTemperature: Measure [0..1]
+ medium: MediumTypeValue [0..1]
+ thermalLossesFactor: Measure [0..1]
+ storageVolume: Volume [0..1]
«FeatureType»
PowerDistributionSystem
«Property»
+ current: Measure [0..1]
+ voltage: Measure [0..1]
«FeatureType»
PowerStorageSystem
«Property»
+ batteryTechnology: CharacterString [0..1]
+ powerCapacity: Measure [0..1]
«enumeration»
MediumTypeValue
air
steam
water


class Timeseries
AbstractFeature
«FeatureType»
Dynamizer::AbstractTimeseries
«Property»
+ firstTimestamp: TM_Position [0..1]
+ lastTimestamp: TM_Position [0..1]
+ adeOfAbstractTimeseries: ADEOfAbstractTimeseries [0..*]
+timeseries
«Property»
1
0..1
«DataType»
TimeseriesComponent
«Property»
+ repetitions: Integer
+ additionalGap: TM_Duration [0..1]
(from Dynamizer)
+component
«Property»
1..*
{ordered}
«FeatureType»
Dynamizer::AbstractAtomicTimeseries
«Property»
+ observationProperty: CharacterString
+ uom: CharacterString [0..1]
+ adeOfAbstractAtomicTimeseries: ADEOfAbstractAtomicTimeseries [0..*]
«FeatureType»
Dynamizer::CompositeTimeseries
«Property»
+ adeOfCompositeTimeseries: ADEOfCompositeTimeseries [0..*]
«FeatureType»
RegularTimeseries
«Property»
+ timeInterval: TM_IntervalLength
+ values: MeasureList
«FeatureType»
Dynamizer::TabulatedFileTimeseries
«Property»
+ fileLocation: URI
+ fileType: TabulatedFileTypeValue
+ mimeType: MimeTypeValue [0..1]
+ valueType: TimeseriesTypeValue
+ numberOfHeaderLines: Integer [0..1]
+ fieldSeparator: CharacterString
+ decimalSymbol: Character [0..1]
+ idColumnNo: Integer [0..1]
+ idColumnName: CharacterString [0..1]
+ idValue: CharacterString [0..1]
+ timeColumnNo: Integer [0..1]
+ timeColumnName: CharacterString [0..1]
+ valueColumnNo: Integer [0..1]
+ valueColumnName: CharacterString [0..1]
+ adeOfTabulatedFileTimeseries: ADEOfTabulatedFileTimeseries [0..*]
«FeatureType»
Dynamizer::GenericTimeseries
«Property»
+ valueType: TimeseriesTypeValue
+ adeOfGenericTimeseries: ADEOfGenericTimeseries [0..*]
+timeValuePair
«Property»
1..*
«DataType»
TimeValuePair
«Property»
+ timestamp: TM_Position
+ intValue: Integer [0..1]
+ doubleValue: Real [0..1]
+ stringValue: CharacterString [0..1]
+ geometryValue: GM_Object [0..1]
+ uriValue: URI [0..1]
+ boolValue: Boolean [0..1]
+ implicitGeometryValue: ImplicitGeometry [0..1]
+ appearanceValue: AbstractAppearance [0..1]
(from Dynamizer)
«DataType»
ADEOfAbstractTimeseries
(from Dynamizer)
«DataType»
TimeValuesProperties
«Property»
+ acquisitionMethod: AcquisitionMethodValue
+ interpolationType: InterpolationTypeValue
+ qualityDescription: CharacterString [0..1]
+ source: CharacterString [0..1]
+ thematicDescription: CharacterString [0..1]
«enumeration»
Dynamizer::
TimeseriesTypeValue
int
double
string
geometry
uri
bool
implicitGeometry
appearance
«DataType»
ADEOfCompositeTimeseries
(from Dynamizer)
«DataType»
PeriodOfYear
«Property»
+ period: TM_Period
«enumeration»
DayTypeValue
monday
tuesday
wednesday
thursday
friday
saturday
sunday
designDay
weekDay
weekEnd
typicalDay
«DataType»
ADEOfAbstractAtomicTimeseries
(from Dynamizer)
«DataType»
DPSProperties
«Property»
+ dayType: DayTypeValue [0..1]
+ period: TM_Period [0..1]
«enumeration»
InterpolationTypeValue
averageInPrecedingInterval
averageInSucceedingInterval
constantInPrecedingInterval
constantInSucceedingInterval
continuous
discontinuous
instantaneousTotal
maximumInPrecedingInterval
maximumInSucceedingInterval
minimumInPrecedingInterval
minimumInSucceedingInterval
precedingTotal
succeedingTotal
«DataType»
ADEOfTabulatedFileTimeseries
(from Dynamizer)
«DataType»
TabulatedFileTimeseriesExtension
«Property»
+ recordSeparator: CharacterString [0..1] = \n
+ timeInterval: TM_IntervalLength [0..1]
«enumeration»
AcquisitionMethodValue
measurement
simulation
calibratedSimulation
estimation
unknown


class Schedule
AbstractFeature
«FeatureType»
Core::AbstractFeatureWithLifespan
«FeatureType»
AbstractSchedule
«FeatureType»
Core::AbstractDynamizer
+dynamizer
«Property»
1
«FeatureType»
ConstantValueSchedule
«Property»
+ averageValue: Measure
«FeatureType»
DualValueSchedule
«Property»
+ usageHoursPerDay: Decimal [0..1] = 24
+ usageDaysPerYear: Decimal [0..1] = 365
+ usageValue: Measure
+ idleValue: Measure
«FeatureType»
DailyPatternSchedule
«Property»
+ timeDependingValues: Decimal
timeDependingValues
requires to be referenced
by Dynamizer
«FeatureType»
TimeseriesSchedule
«Property»
+ timeDependingValues: Decimal
timeDependingValues
requires to be referenced
by Dynamizer


class WeatherData
«enumeration»
WeatherDataTypeValue
airTemperature
humidity
windSpeed
cloudiness
globalSolarIrradiance
directSolarIrradiance
diffuseSolarIrradiance
terrestrialEmission
downwardTerrestrialRadiation
daylightIlluminance
«DataType»
ADEOfAbstractCityObject
(from Core)
«DataType»
EnergyADECityObjectProperties
AbstractPhysicalSpace
«FeatureType»
Core::AbstractOccupiedSpace
«FeatureType»
WeatherStation
«Property»
+ stationName: CharacterString [0..1]
+weatherData
«Property»
0..*
values requires to be
referenced by Dynamizer
AbstractFeatureWithLifespan
«FeatureType»
WeatherData
«Property»
+ weatherDataType: WeatherDataTypeValue
+ values: Measure
+position
«Property»
0..1
GM_Primitive
«type»
Geometric primitive::
GM_Point
+dynamizer
«Property»
1
AbstractFeatureWithLifespan
«FeatureType»
Core::AbstractDynamizer