International Journal of Geo-Information

MDPI

Article

Mapping the CityGML Energy ADE to CityGML 3.0 Using a Model-Driven Approach

Carolin Bachert [1], Camilo León-Sánchez [2], Tatjana Kutzner [3] and Giorgio Agugiaro [2,*]

[1] con terra GmbH, Martin-Luther-King-Weg 20, 48155 Münster, Germany; c.bachert@conterra.de
[2] 3D Geoinformation Group, Department of Urbanism, Faculty of Architecture and the Built Environment, Delft University of Technology, Julianalaan 134, 2628 BL Delft, The Netherlands; c.a.leonsanchez@tudelft.nl
[3] Chair of Geoinformatics, Technical University of Munich, Arcisstr. 21, 80333 München, Germany; kutzner@tum.de
* Correspondence: g.agugiaro@tudelft.nl

Abstract: With the increasing adoption of semantic 3D city models, the relevance of applications in the field of Urban Building Energy Modelling (UBEM) has rapidly grown, as the building sector accounts for a large part of the total energy consumption. UBEM allows us to better understand the energy performance of the building stock and can contribute to defining refurbishment strategies. However, UBEM applications require lots of heterogeneous data, eventually advocating for standards for data interoperability. The Energy Application Domain Extension has been created to cope with UBEM data requirements and offers a standardised data model that builds upon the CityGML standard. The Energy ADE 1.0, released in 2018, creates new classes and adds new properties to existing classes of the CityGML 2.0 Core and Building modules. CityGML 3.0, released in 2021, has introduced several changes to the data model and its ADE mechanism. These changes render the Energy ADE incompatible with CityGML 3.0. This article presents how the Energy ADE has been ported to CityGML 3.0 to allow, on the one hand, for a lossless data conversion and, on the other hand, to exploit the new characteristics of CityGML 3.0 while keeping a logical symmetry between the ADE classes of both CityGML versions. The article describes the chosen methodology, the mapping strategies, the implementation steps, as well as the data conversion tests to check the validity of the "new" Energy ADE for CityGML 3.0.

Keywords: CityGML; Energy ADE; model-driven mapping; data modelling; UBEM

Citation: Bachert, C.; León-Sánchez, C.; Kutzner, T.; Agugiaro, G. Mapping the CityGML Energy ADE to CityGML 3.0 Using a Model-Driven Approach. *ISPRS Int. J. Geo-Inf.* **2024**, *13*, 121. https://doi.org/10.3390/ijgi13040121

Academic Editors: Sisi Zlatanova and Wolfgang Kainz

Received: 5 February 2024
Revised: 19 March 2024
Accepted: 30 March 2024
Published: 4 April 2024

1. Introduction

Standardised data models can play a vital role in areas where complex information is handled by various stakeholders coming from different backgrounds as they ensure lossless data exchange, facilitate the development of reliable software solutions and, therefore, enhance the overall data interoperability.

Urban Building Energy Modelling (UBEM) represents a good example in this regard. In UBEM, different scenarios of the energy demand and supply of a city can be simulated at the individual building level [1]. As such, UBEM serves as a valuable set of approaches, methods, and tools to support decision-makers in detecting energy-saving potentials and in subsequently allocating required resources for retrofitting purposes [2]. However, UBEM requires large and heterogeneous quantities of information, such as data regarding energy consumption, local climate, occupant behaviour, physical properties of the buildings and their geometries [3,4].

The international standard City Geography Markup Language (CityGML), issued by the Open Geospatial Consortium (OGC), offers the possibility to model urban environments, including buildings, in a 3D space. CityGML defines "basic entities, attributes, and relations" of relevant urban objects, functioning both as a conceptual data model for

semantically enriched 3D city models and as a storage and exchange format [5]. Currently, the most widely used version is CityGML 2.0, which was released in 2012.

CityGML is intentionally designed to be application-independent. Nevertheless, in certain cases, additional classes or attributes may be needed for specific domains. For this reason, CityGML can be extended following two approaches. The first one allows for the definition of so-called generic attributes and generic city objects without the need to extend the conceptual data model. The second one, on the other hand, offers more modelling capabilities but demands an extension of the data model. This second approach is referred to as the Application Domain Extension (ADE) mechanism.

The Energy ADE version 1.0 is such an extension for CityGML. It builds upon the CityGML 2.0 Core and Building modules and extends them by means of additional classes and properties. As such, it has been conceived and designed as a solution to model and store relevant data needed for UBEM. It offers the possibility to model both data serving as input for energy-related applications and data storing the application results, in order to facilitate further building or city-wide energy assessments. The ultimate goal is to promote data interoperability between different UBEM stakeholders by means of a standardised data model [6]. The Energy ADE 1.0 was released in 2018 through a joint effort of various international parties and stakeholders (familiar/expert with/in either CityGML or UBEM). It is mentioned in the literature as a best-practice example when it comes to ADE development due to its technical maturity and available documentation. As a result, the Energy ADE has already been used in several national and international projects [7,8].

In September 2021, version 3.0 of the CityGML standard was released by the OGC. The new version introduces considerable changes to the Core module with a revised geometry concept and a newly established space concept. Furthermore, new modules for time-dependent properties and man-made constructions have been added. Moreover, it is easier to include several ADEs at once thanks to an improved ADE mechanism [7].

These changes directly affect the portability of the Energy ADE to CityGML 3.0. First, the Energy ADE is not compatible with the revised ADE mechanism. Second, some classes have been changed in terms of name, attributes, or overall hierarchical position in the data model. Thus, they do not link seamlessly to the existing Energy ADE data model anymore. Furthermore, some classes and properties in the Energy ADE are now already natively incorporated in CityGML 3.0. This makes certain Energy ADE classes obsolete or redundant—which is against the main raison d'être of an ADE, i.e., to extend the data model only where necessary. Finally, CityGML 3.0 introduces many additional classes, which potentially represent a better semantic fit from which to derive ADE classes.

The Energy ADE covers a variety of technical aspects and is a good example of how ADEs can incorporate and take advantage of new functionalities of CityGML 3.0. Additionally, there are currently no well-documented and published examples of *existing* ADEs being ported from CityGML 2.0 to CityGML 3.0. For this reason, this article presents the work carried out to map the Energy ADE to CityGML 3.0. The challenge and the goal have been to map the Energy ADE classes, wherever and whenever necessary, to the CityGML 3.0 data model, without any losses in terms of semantics and functional modelling capabilities. Eventually, data modelled according to CityGML 2.0 and Energy ADE 1.0 must fulfil the requirement to be convertible to CityGML 3.0 extended with the "new" Energy ADE and without any data losses. The conceptual work carried out in this process and the main implementation steps will be presented, including the data conversion and tests carried out to evaluate the mapping. Particular attention is paid to establishing a uniform, rule-based mapping and its reasoning.

Keeping these overall goals in mind, the core of the investigation has been dedicated to understanding to which extent the Energy ADE for CityGML 2.0 needs to be adapted in order to be conformant with the newly released CityGML 3.0 standard. Throughout this process, the Energy ADE 1.0 classes that become obsolete, those that need to be adapted, and those that can be mostly taken over have been identified. The resulting "new" Energy ADE

for CityGML 3.0 is available both as a UML class diagram and as an XML schema definition (XSD) file. Lastly, data conversion tests are carried out using Safe Software's FME.

When it comes to CityGML 2.0, a lot of experience has already been reported in the past decade regarding the creation of ADEs. Biljecki et al. [8] provide an extensive review of many heterogeneous ADEs created for different application domains. Additionally, a formal UML-based approach to create ADEs has been proposed by van den Brink et al. [9] which is based on the well-established model-driven approach and has also been applied to the development of CityGML 3.0 [10].

However, due to the relatively recent publication of the CityGML 3.0 conceptual model, not much literature has been published up to now regarding ADEs for CityGML 3.0 or the mapping of existing ADEs to the new CityGML version.

Biljecki et al. [11] propose how to extend CityGML 3.0 in order to convert data from IFC to CityGML 3.0. Starting from the awareness that differences in the scope and intent between IFC and CityGML lead to inevitable data losses when converting the former to the latter, the authors identify a subset of IFC data that is beneficial to keep and convert to CityGML by means of an ad hoc ADE.

In the context of Underground Land Administration (ULA), Seidian et al. [12,13] have recently dealt with the modelling of underground legal boundaries, in order to tackle the lack of a link between underground physical and legal data in current practices. As a result, they propose to extend CityGML 3.0 by means of the so-called VicULA (Victoria Underground Land Administration) ADE, in which underground legal data elements can be logically embedded into a 3D data model. The VicULA ADE has been specifically developed for Victoria, Australia, however, according to the authors, the proposed model and approach can be used and replicated in other jurisdictions by adjusting the data requirements for underground legal boundaries.

The Utility Network ADE [14] represents a valuable source of inspiration for mapping an existing ADE from CityGML 2.0 to 3.0—which best resembles the core of the work presented in this article. The Utility Network ADE is openly available on GitHub not only as a UML diagram. It also includes the derived XSD schema and the accompanying configuration files to carry out the conversion using ShapeChange (more details will be provided later in the article). However, unfortunately, neither scientific publications nor detailed documentation of the process is currently available.

Finally, Bachert [15] has recently and specifically dealt with the conversion of the Energy ADE from CityGML 2.0 to 3.0. This article is extracted from and extends this work. Therefore, as a result of the relative scarcity of available publications, this article can represent a reference for other existing ADEs to be converted to CityGML 3.0.

The following sections will provide more details on each of the above-mentioned steps. The article is structured as follows: Section 2 describes the applied method, followed in Section 3 by some theoretical background on the Energy ADE and the description of some updates in CityGML 3.0 that are relevant to this article. In this context, we would like to point out that we assume that the reader is already familiar with the general concepts of CityGML 2.0, CityGML 3.0, and UML modelling. The same applies to the Energy ADE, for which we only provide an overview of its main characteristics while referencing further existing literature for the reader who may want (or need) to read more extensively about it.

Section 4 contains a detailed explanation of the mapping and its logic, followed by further steps of the implementation in Section 5. Section 6 presents and discusses the result and Section 7 contains the conclusions, as well as the outlook.

Although several UML excerpts from the Energy ADE, CityGML 2.0 and CityGML 3.0 are provided throughout the article, we nevertheless heartily advise the reader to have the full UML class diagrams at hand. They can be retrieved at the following links:

- Energy ADE 1.0: https://www.citygmlwiki.org/index.php/CityGML_Energy_ADE (accessed on 21 February 2024)
- CityGML 2.0 and 3.0: https://www.ogc.org/standard/citygml (accessed on 21 February 2024).

2. Methodology Overview

The work presented in the article follows and adapts the UML-based approach to create ADEs by van den Brink et al. [9]. The creation of the "new" Energy ADE is complemented with the actual transformation of test data from CityGML 2.0 + Energy 1.0 to CityGML 3.0 + the "new" Energy ADE, in order to test and verify the conversion from one data model to the other without any data losses.

Overall, the developed methodology consists of three steps which are summarised in Figure 1. Following the above-mentioned model-driven approach, first, a data model is defined at the conceptual level including its required classes, properties and relations. For this purpose, UML is chosen as the formal modelling language to define the mapped ADE. The conceptual mapping process constitutes the core of this work and is first carried out module by module in a "pen-and-paper" approach. Only afterwards, the UML class diagrams are created using the modelling software Enterprise Architect v. 13. This first step is shown in Figure 1 in dark transparent green.

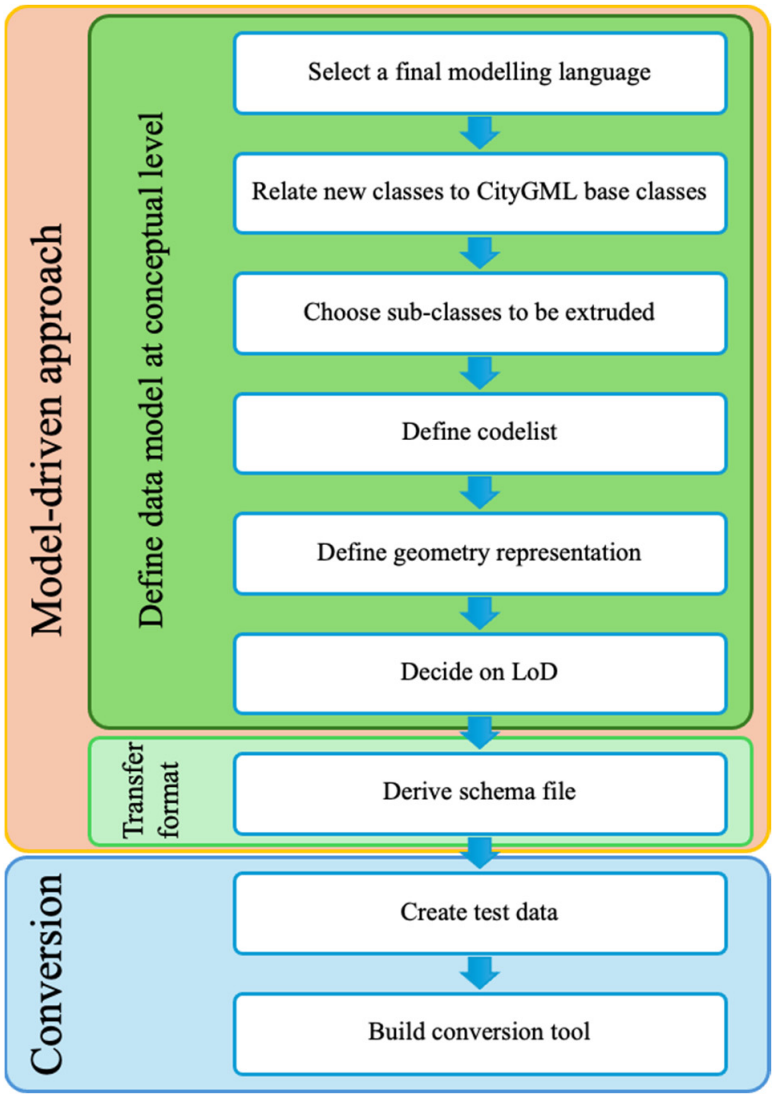

Figure 1. Schematic workflow to map and convert the Energy ADE to CityGML 3.0. Image adapted from [9].

In the second step, depicted in Figure 1 in light transparent green, the transfer format is derived from the UML data model. Here, based on the GML target encoding, an XSD schema file is derived. It specifies how to correctly read, write and validate Energy ADE GML files for CityGML 3.0. The XSD schema is created using the Java tool ShapeChange v. 2.11, which requires a specified configuration file and then automatically applies the encoding rules to the UML class diagrams.

The last, third step, depicted in light transparent blue in Figure 1, consists of creating a CityGML 2.0 + Energy ADE test dataset which is then converted to a CityGML 3.0 + "new" Energy ADE dataset. This third step is meant to test the overall applicability of the newly mapped ADE and to prove whether data can be indeed converted without data losses. Therefore, the test dataset has been intentionally prepared to cover every feature type, property and relation of the Energy ADE at least once. Both the dataset creation and the conversion are implemented with the ETL software FME Desktop v. 2022. The conversion workspace builds upon a pre-existing template available on the FME Hub, which converts the Building module and other frequently used classes to CityGML 3.0 [16]. The methodology briefly presented here will be described in more detail in the coming sections.

3. Theoretical Background

This section provides the most important background knowledge regarding the Energy ADE for CityGML 2.0 and the changes in CityGML 3.0 that are relevant in this context. The goal is thereby not to cover every aspect of these data models, but to focus only on those that are most important to understand the mapping process.

3.1. Energy ADE 1.0 for CityGML 2.0 in a Nutshell

The Energy ADE builds upon the CityGML 2.0 Core and Building modules. It consists of six thematic modules in which either new classes or classes extending CityGML classes are defined, together with additional data types, several codelists, and enumerations. Figure 2 shows the overview of the ADE packages and depicts the dependencies between the different modules. The thematic modules are briefly described below, whereas a more detailed explanation is given alongside the selected examples in Section 4.

- The **Energy ADE Core** module (in light pink) defines additional attributes for the CityGML *Building::_AbstractBuilding* and CityGML *Core::_CityObject* classes. It also provides new abstract base classes for the other modules and establishes additional data types and enumerations;
- The **Occupant behaviour** module (in light green) 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.;
- The **Material and construction** module (in blue) enables the modelling of the composition of construction surfaces through different layers and their physical properties;
- The **Energy systems** module (in orange) provides classes to model the energy storage, distribution, emission, and conversion systems of a building and interrelates them to represent the respective energy exchange;
- The **Building physics** module (in light yellow) defines classes for thermal zones, thermal boundaries and thermal openings to model the thermal hull of a building (or subparts of it);
- The **Supporting classes** module (in yellow) comprises classes for different schedules and time series. They are used to add time-dependent values to the other module parameters. Additionally, a *WeatherStation* class is defined herein.

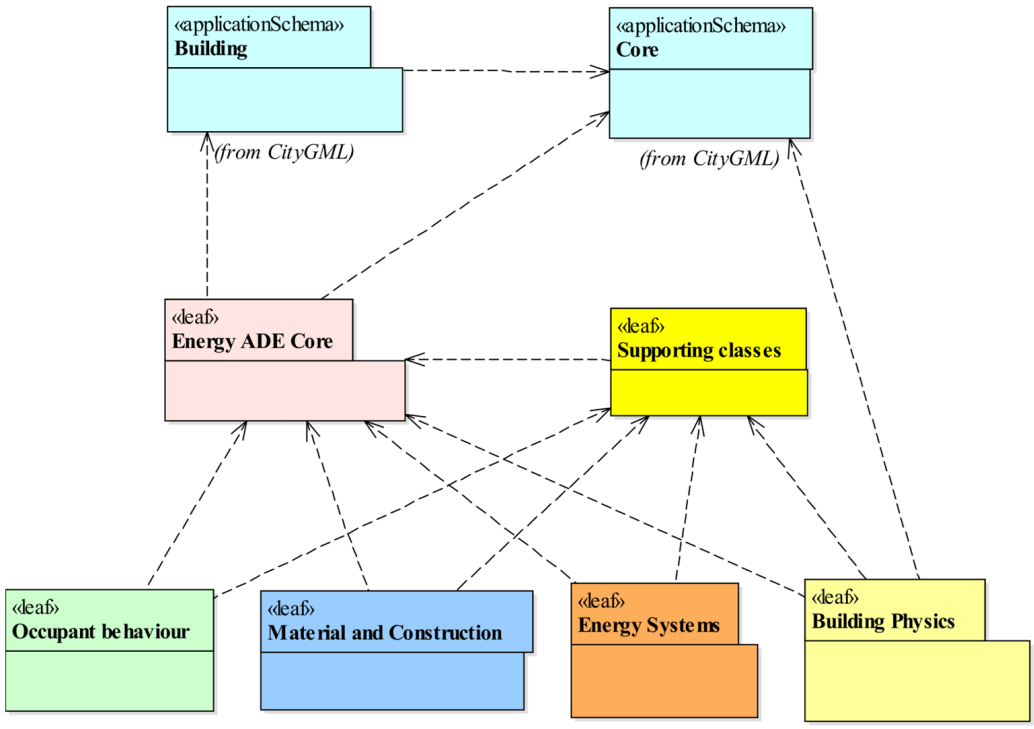

Figure 2. Package overview of the Energy ADE 1.0 for CityGML 2.0. The colours representing the different packages will be used throughout the article.

A complete description of the Energy ADE, its overall structure, as well as its classes is provided by Agugiaro et al. [6]. Please note that the colours presented in the package diagram depicted in Figure 2 are also adopted in the remainder of the article for better readability. Classes belonging to the ADE modules are represented using the same aforementioned colours, while all classes used for CityGML (both version 2.0 and 3.0) are always depicted in cyan.

3.2. Relevant Changes in CityGML 3.0

CityGML 3.0 comes with several changes intended to increase its suitability for various user groups and expand its range of potential applications in fields such as urban planning, energy and environment simulations, traffic analyses, Internet of Things, and Smart Cities. Overall, the revisions in CityGML 3.0 can be categorised into five aspects. First, the standard applies a model-driven approach, i.e., it is now formally defined through a platform-independent Conceptual UML Model from which various exchange formats can automatically be derived. Second, as seen in Figure 3, new modules are introduced (Construction, Versioning, Dynamizer, PointCloud) and existing ones are revised (Generics, Core, Building, Transportation). These new modules allow the representation of the dynamic behaviour of city models, to model the transportation infrastructure and constructions in more detail and to represent the geometries of city objects by 3D point clouds. Third, there is a newly introduced space concept. All geometries, including an updated LOD concept, are defined now in the Core module. Fourth, a refined ADE mechanism simplifies the inclusion of several ADEs simultaneously and, furthermore, supports their creation based on the model-driven approach [10]. Finally, the interoperability with the European Union directive INSPIRE, as well as with various other standards such as IndoorGML and IFC and with linked data and Semantic Web Technologies such as RDF, was improved.

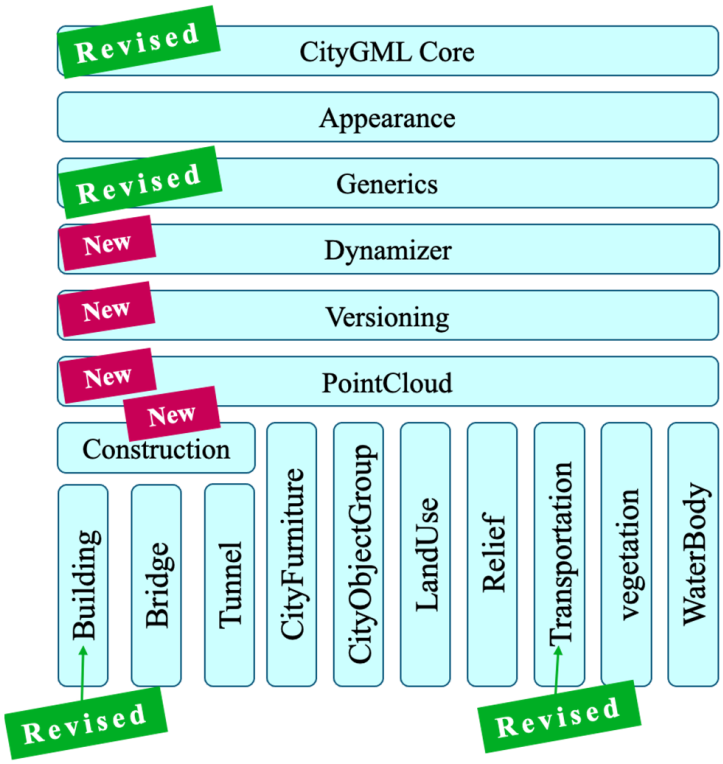

Figure 3. The modules in CityGML 3.0. Image adapted from [10].

Since the new space concept and the adapted ADE mechanism play an important role throughout this research, they are explained here in more detail. For an in-depth introduction to the other changes in CityGML 3.0 please refer to Kutzner et al. [10].

3.2.1. Space Concept

In the Core module, new abstract classes defining different notions of space are introduced. All city objects now derive directly or indirectly from one of these abstract classes, adding an additional level of semantic meaning. An overview of the classes and their relation is given in Figure 4.

First, every city object is distinguished based on whether it is of volumetric (*AbstractSpace*) or areal extent (*AbstractSpaceBoundary*). With an *AbstractSpace* class, real-world volumetric objects can be modelled, whereas an *AbstractSpaceBoundary* class describes objects which bound or delimit volumetric objects from each other, e.g., wall surfaces (Figure 5).

One level further, *AbstractSpace* is subdivided into *AbstractLogicalSpace* and *AbstractPhysicalSpace*. The latter represents physically tangible objects which are "fully or partially bounded by physical objects" such as buildings bounded by walls and a roof [10]. On the contrary, an *AbstractLogicalSpace* is an entity defined by a thematic meaning that can also have a virtual boundary. Examples can be abstract such as a traffic zone, or more tangible such as an apartment consisting of several physical spaces (Figure 6).

Lastly, *AbtractPhysicalSpace* is further subclassed into *AbstractUnoccupiedSpace* and *AbstractOccupiedSpace*. *AbstractOccupiedSpace* describes volumetric objects which prevent the placement of other city objects at that place. Consequently, an *AbstractUnoccupiedSpace* object models volumetric objects that are free to put other things in or to walk through [10]. The example of a building, as seen in Figure 7, helps to illustrate this concept. The building as a whole constitutes an *AbstractOccupiedSpace* as nothing else can be placed in this specific space anymore. In turn, rooms within the building are empty volumetric objects and thus *AbstractUnoccupiedSpaces*. Furniture placed inside the rooms occupies space and is therefore modelled as an *AbstractOccupiedSpace*.

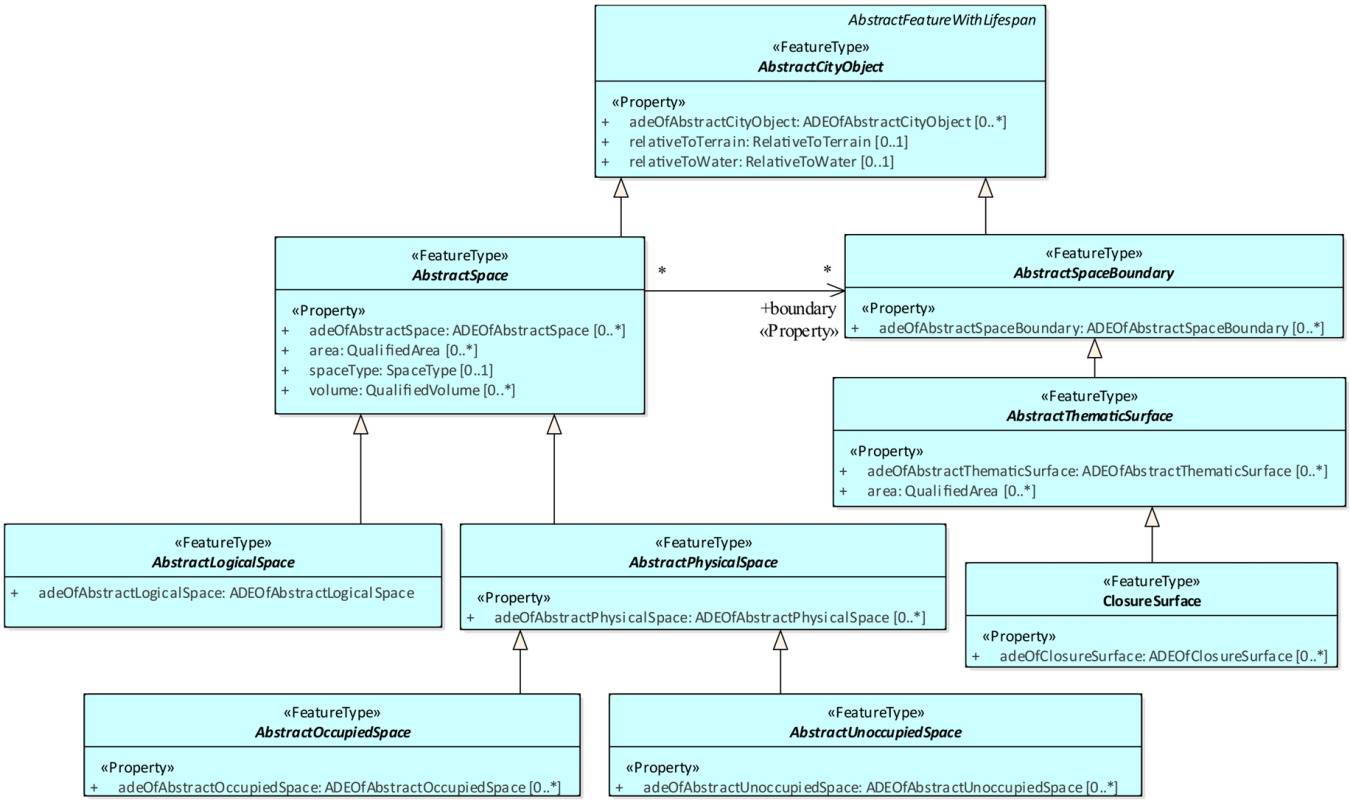

Figure 4. Overview of the classes making up the space concept in the CityGML 3.0 Core module. Image adapted from [7].

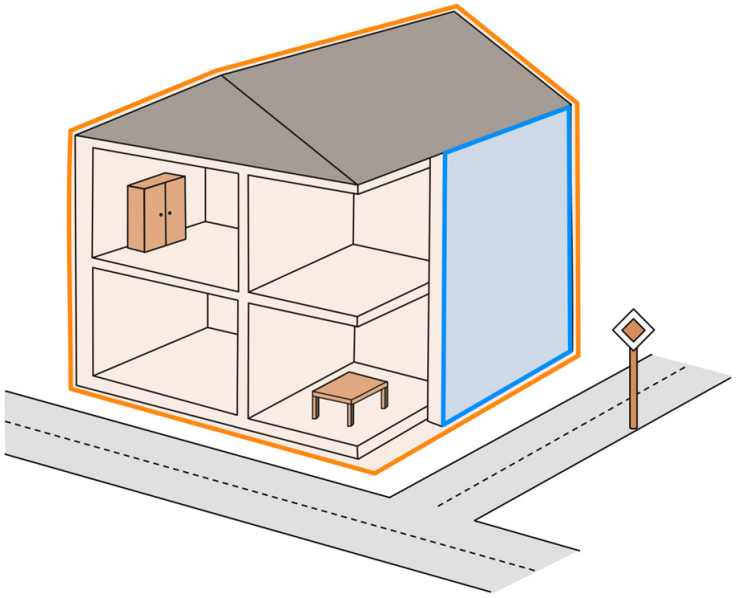

Figure 5. Representation of the classes *AbstractSpace* (orange) and *AbstractSpaceBoundary* (blue) using the example of a building. Image adapted from [10].

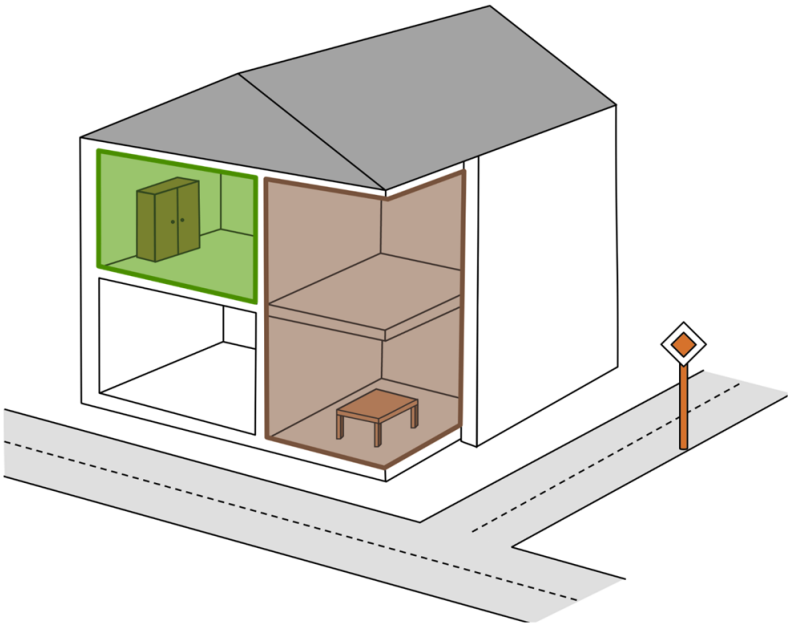

Figure 6. Representation of the classes *AbstractPhysicalSpace* (green) and *AbstractLogicalSpace* (brown) using the example of a building. Image adapted from [10].

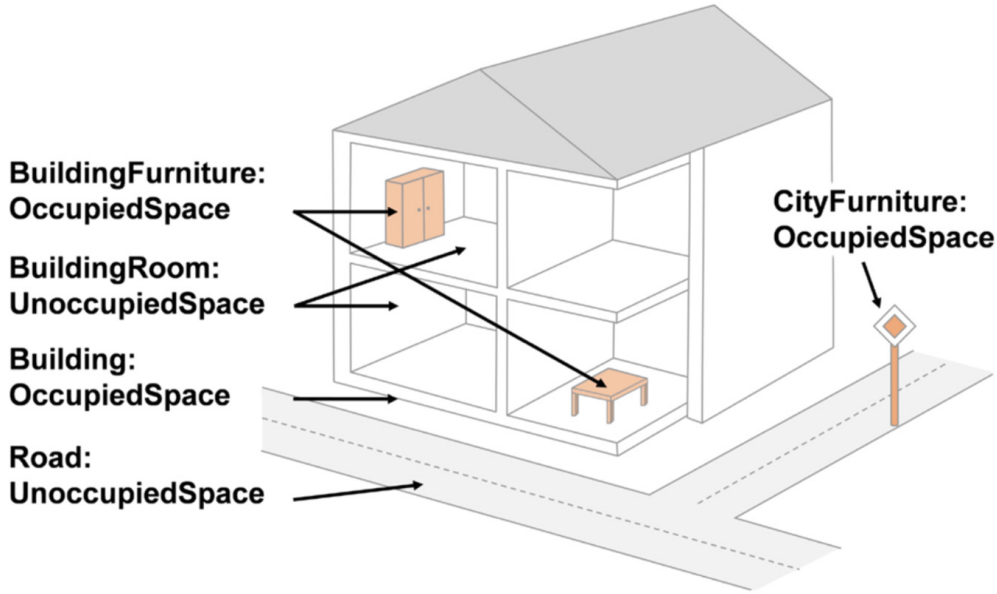

Figure 7. Representation of the classes *OccupiedSpace* and *UnoccupiedSpace* using the example of a building. Image taken from [10].

3.2.2. ADE Mechanism

As in previous versions of the standard, the CityGML 3.0 data model can be extended by means of ADEs. New is that they now have to be defined through UML class diagrams in order to be encoding-independent. Moreover, the ADE hook mechanism has been redesigned in order to facilitate the integration of multiple ADEs at once [7].

ADEs allow the extension of the CityGML data model in two ways. Both of them are depicted in Figure 8 using examples. The first one introduces new classes by deriving them from *AbstractFeature* (or as shown in the figure, from a semantically fitting subclass of *AbstractFeature* such as *AbstractLogicalSpace*). As such, the extension mechanism through specialisation classes remains the same as in CityGML 2.0.

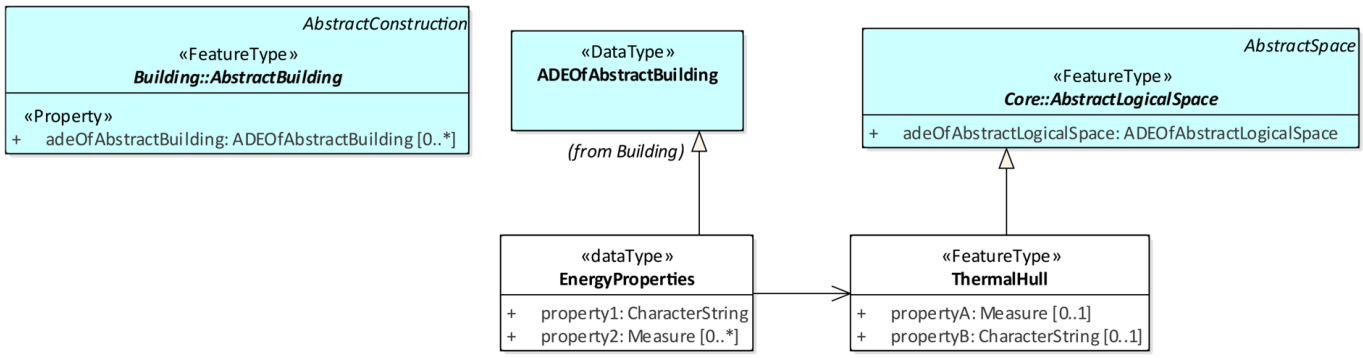

Figure 8. Example of extending the existing CityGML 3.0 class *AbstractBuilding* by means of the ADE hook mechanism (*EnergyProperties*, via *ADEOfAbstractBuilding*) and by deriving a new class (*ThermalHull*) from the existing class *AbstractLogicalSpace*.

Using the second possibility, also referred to as the ADE hook mechanism, additional properties can be added to existing CityGML classes. The way to do this has been updated so that subclassing the respective CityGML classes is not necessary anymore, as was the case in CityGML 2.0. Every CityGML class has now an attribute "adeOf*FeatureTypeName*" of type "ADEOf*FeatureTypeName*", with *FeatureTypeName* being replaced by the corresponding CityGML class name (e.g., *adeOfWallSurface* of type *ADEOfWallSurface*). The new properties are injected into the CityGML class by subclassing the corresponding data type "ADEOf*FeatureTypeName*". In the example of Figure 8, *ThermalHull* is defined as a new class derived from *AbstractLogicalSpace* and the *EnergyProperties* data type defines new properties for the class *AbstractBuilding*.

4. Mapping the Energy ADE to CityGML 3.0

As described in Section 2, the methodology to map the Energy ADE to CityGML 3.0 consists of three main steps. This section describes the first step which comprises the detailed mapping process. In order to do so, rules that generally apply throughout all modules are defined beforehand. Section 5 will further elaborate on the remaining two steps, namely the corresponding XSD file derivation and the creation of test data based on CityGML 2.0 and Energy ADE 1.0, as well as its conversion to CityGML 3.0 + "new" Energy ADE.

The goal is to perform the mapping without changing the contents of the Energy ADE and, thus, to convey the same information as before. However, in order to ensure logical consistency and a coherent modelling style throughout all modules, a set of mapping guidelines has been established.

4.1. Mapping Principles

The *general mapping principles* provide general instructions on how classes should be mapped, especially if there are several alternative possibilities. On the other hand, the *overarching mapping decisions* are more concrete. They are distinctive mapping rules which apply to all ADE classes and/or relations.

When it comes to the *general mapping principles*, they are:

- "Integrate as much as possible": According to this principle, Energy ADE classes should be integrated as "deep" into the CityGML 3.0 UML model as possible. This allows for the use of the new space and geometry concept and, thus, adds another layer of semantic meaning to the classes. In addition, the ADE classes do not need to define their own geometries anymore and can benefit from a multiple LOD representation, inherited directly from the CityGML 3.0 Core module. An alternative would be to keep the ADE classes closer to each other at a very high level in the UML model (i.e., deriving them, for example, directly from *AbstractFeature*). However, as a consequence,

the geometry and space concept would not apply to them—which in fact would disregard one of the main changes in CityGML 3.0.

- "Maintain logical symmetry": This principle suggests that classes that are similar in the Energy ADE should be mapped in a similar way to CityGML 3.0 in order to obtain a logically consistent mapping. For example, ADE classes with a similar meaning, or at the same conceptual level, should be mapped to the same hierarchy level or be derived from the same parent class in the CityGML 3.0 data model.

On the individual level, the integration of an Energy ADE class into CityGML 3.0 depends on various factors. The primary factor is the compatibility between the ADE class and its potential parent class. An ADE class might fit multiple CityGML 3.0 classes within their specialisation path (e.g., *AbstractBuilding* or *Building*). In such cases, it has to be assessed whether additional properties and relationships of the more specialised class add value to the ADE class, or not. Additionally, a comparison is made on how similar classes are mapped to fulfil the second general modelling principle. It is also necessary to examine whether the decision might inadvertently impact other ADE classes, such as by introducing properties inherited by another ADE class.

While, in the first moment, these mapping principles may seem abstract, they will become clearer through the provided examples later on. It is however essential to note that these principles allow flexibility, sometimes offering multiple solutions in specific scenarios. Eventually, the decision is made at the level of the individual classes. Nonetheless, some specific overarching mapping decisions account for all classes and are summarised in the following list.

When it comes to the *overarching mapping decisions*, they are:

- "*AbstractFeatureWithLifespan* over *AbstractFeature*": *AbstractFeatureWithLifespan* is always preferred as the parent class over *AbstractFeature*. This allows for the inclusion of properties such as *validFrom* and *validTo*. Therefore, every ADE object can be depicted in various versions across its historical timeline;
- "Maintain abstract classes": Abstract classes enable the modular structure of UML class diagrams and facilitate a clear connection between the different modules. On top of that, they are kept for symmetry reasons between the original and the "new" Energy ADE;
- "Keep multiplicities, relations and properties": The multiplicities, relations and properties remain as they are unless there is a specific reason to change them in the "new" Energy ADE version.

The following part demonstrates by means of examples how the Energy ADE is concretely mapped to CityGML 3.0 with explanations of the reasoning behind it. Due to the size of the Energy ADE, the given cases cover only the most important aspects and particularities of the mapping. However, the detailed full mapping can be found in [15]. The examples are organised by modules and are always preceded by a brief explanation of how the module is defined in the Energy ADE for CityGML 2.0.

4.2. The Core Module in the Energy ADE for CityGML 2.0

The Core module, depicted in Figure 9, extends the CityGML abstract classes *_CityObject* and *_AbstractBuilding*. The CityGML 2.0 class *_AbstractBuilding* is extended by means of the ADE hook mechanism to include properties needed for the computation of the building energy demand. This includes attributes regarding its geometry (e.g., *volume*, *floorArea*), construction typology (*constructionWeight*) and energy archetype of building (*buildingType*). Additionally, information regarding *WeatherData* or *EnergyDemand* can be associated with every *_CityObject*. *WeatherData* information is needed either to perform accurate simulations or to store the pre-computed weather-related information (e.g., from solar irradiation pre-processing). *EnergyDemand*, on the other hand, is used to describe an object's time-dependent energy demand, be it in terms of electricity, (natural) gas, etc. Additionally, the Core module establishes anchor points to the remaining ADE modules by means of other abstract classes (e.g., *AbstractThermalZone*, *AbstractUsageZone*) and displays

their interrelations. Finally, it defines new base classes for the remaining modules and introduces new enumerations and codelists.

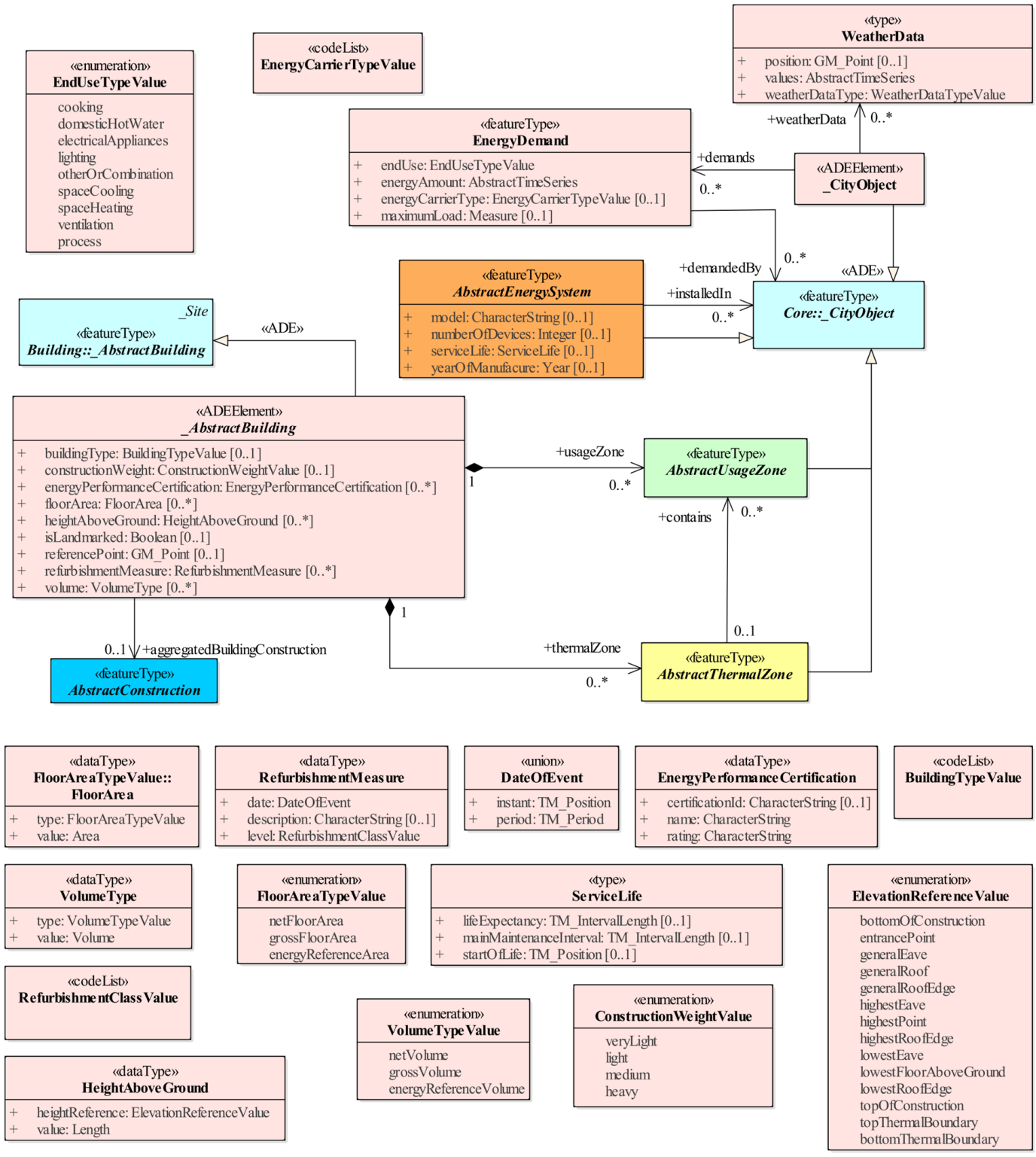

Figure 9. The Core module of the Energy ADE for CityGML 2.0.

4.3. Mapping the Core Module to CityGML 3.0

4.3.1. BuildingProperties

As previously mentioned, additional properties are injected into *AbstractBuilding* via the ADE hook mechanism. In CityGML 3.0, the new data type *BuildingProperties* is derived from *ADEOfAbstractBuilding* and is used to add the corresponding Energy ADE properties to CityGML 3.0 *AbstractBuilding*. The fully mapped Core module of the Energy ADE for CityGML 3.0 is depicted in Figure 10.

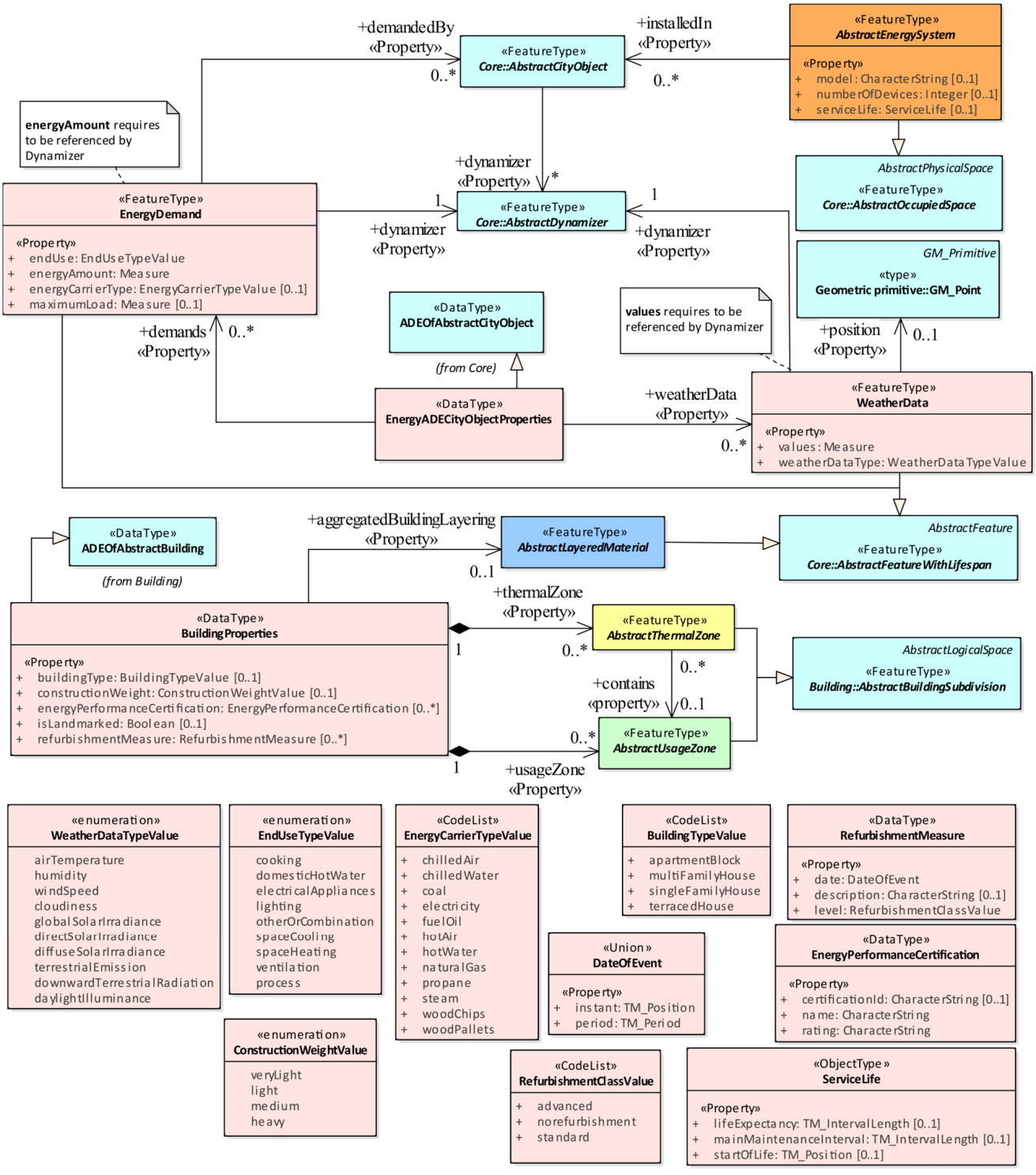

Figure 10. The Core module of the Energy ADE for CityGML 3.0.

Conveniently, some Energy ADE properties can be "replaced" by equivalent ones already provided in CityGML 3.0, i.e., *volume*, *floorArea*, and *heightAboveGround*. How the properties are transferred is shown using the example of *volume* in Figure 11. Likewise, *floorArea* is mapped to the *area* property of *AbstractSpace* and *heightAboveGround* to the property *height* of *AbstractConstruction* in the newly added Construction module of CityGML 3.0. Finally, as CityGML 3.0 centralises all geometries in the Core module, Energy ADE geometry properties such as the *referencePoint* are mapped directly to the already existing *lod0Point* property.

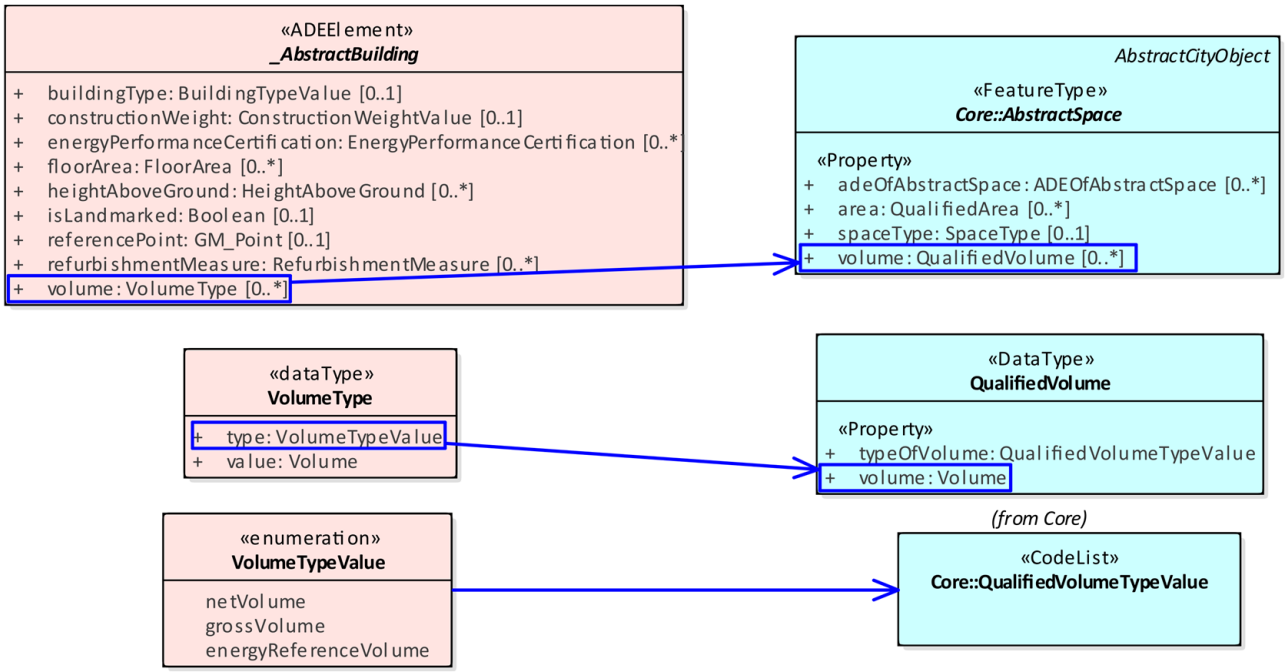

Figure 11. Mapping the volume attribute of *_AbstractBuilding* in the Energy ADE for CityGML 2.0 (on the left) to the volume attribute of *AbstractSpace* in CityGML 3.0 (on the right). The corresponding complex data types are matched accordingly.

4.3.2. EnergyDemand

According to the Energy ADE for CityGML 2.0 (see Figure 9) a *_CityObject* (including its specialisation classes) can *demand* multiple *EnergyDemand* instances. However, association relationships cannot be added directly to any CityGML class, as this would alter the original data model. Consequently, the CityGML 2.0 *_CityObject* class itself must be extended by means of the ADE hook, from which the relation to the *EnergyDemand* class can be defined. In this case, the new data type *EnergyADECityObjectProperties* is created for the CityGML 3.0 class *AbstractCityObject*. From here, the relation to *EnergyDemand* is made.

When it comes to the *EnergyDemand* class, the mandatory property *energyAmount* is linked to a further class that is used to model time series. For CityGML 3.0, the Energy ADE *AbstractTimeSeries* class (and its subclasses) is largely integrated into CityGML 3.0's Dynamizer module. Thus, the way how properties are modelled for time-varying properties has changed considerably. Similarly, all Energy ADE classes having a property linked to a time-dependent class now require a relation to *AbstractDynamizer*. CityGML 3.0 already provides a relation from any *AbstractCityObject* to *AbstractDynamizer* (as can be seen in the CityGML 3.0 Core module). Yet, *EnergyDemand* is derived from *AbstractFeatureWithLifespan* and therefore this relation needs to be created additionally. This is achieved by a relation from *EnergyDemand* to *AbstractDynamizer* with the role name *dynamizer*. The multiplicity of 1 makes it a mandatory relation. As the property *energyAmount* itself is obligatory for *EnergyDemand*, the multiplicity of 1 ensures the connection to *AbstractDynamizer* and, thus, serves as a security check for the modelling of time-varying property values. Beyond this,

the specifics of the Dynamizer module and the time series data are explained in further detail later on, in Section 4.9.

4.4. The Building Physics Module in the Energy ADE for CityGML 2.0

According to the Energy ADE, a building can be subdivided into one or several thermal zones (corresponding to class *ThermalZone*), with each zone having its own thermal behaviour. The thermal zones are delimited from each other or the exterior of the building by thermal boundaries (class *ThermalBoundary*). Doors, windows or other openings within the thermal boundary represent thermal openings (class *ThermalOpening*).

Each one of these three classes can be optionally associated with a geometry (a Solid for the *ThermalZone*, and a MultiSurface for *ThermalBoundary* and *ThermalOpening*) to represent their explicit geometry. Please note that, by decision of the Energy ADE designers, such properties (i.e., *volumeGeometry* and *surfaceGeometry*) allow only for a single representation and, thus, are decoupled from the usual LOD representation typical of CityGML [6]. *ThermalZones* can furthermore contain multiple *UsageZone* instances, the respective abstract class *AbstractUsageZone* is depicted in the Core module. Beyond this, the thermal and optical properties of *ThermalOpening* and *ThermalBoundary* can be described through their relation to *AbstractConstruction*. Figure 12 depicts an overview of the Building physics module of the Energy ADE.

4.5. Mapping the Building Physics Module to CityGML 3.0

As already mentioned, there is usually more than one possibility to perform a mapping. The Building physics module is a good example thereof as it also illustrates how the mapping principles presented in Section 4.1 come to fruition.

The first mapping option is to derive the *ThermalZone*, *ThermalBoundary* and *ThermalOpening* classes directly from *AbstractCityObject*, shown in Figure 13. As such, the implementation would be very similar to the one in the Energy ADE for CityGML 2.0. Consequently, the classes stay close together at the same hierarchy level within CityGML 3.0. All properties remain unchanged, and the geometries are explicitly defined within the new classes.

Alternatively, the ADE classes can be integrated deeper into the CityGML 3.0 data model depending on their best semantic fit. For example, the class *ThermalZone* can be subclassed from *AbstractSpace*, while the classes *ThermalBoundary* and *ThermalOpening* become a specialisation of *AbstractSpaceBoundary* (see Figure 14). In this way, some of the Energy ADE properties can be mapped to already existing CityGML 3.0 ones, as seen before in the case of *AbstractBuilding*. Additionally, the geometries do not need to be explicitly defined inside the ADE classes anymore, as they are now inherited from those existing in the CityGML 3.0 Core module. Besides, the space concept in the Core module enriches the ADE classes with an additional level of semantic meaning.

Due to these reasons, the latter mapping approach is the preferred one and is further pursued. In the following, it is discussed how the three Energy ADE classes are modelled in detail.

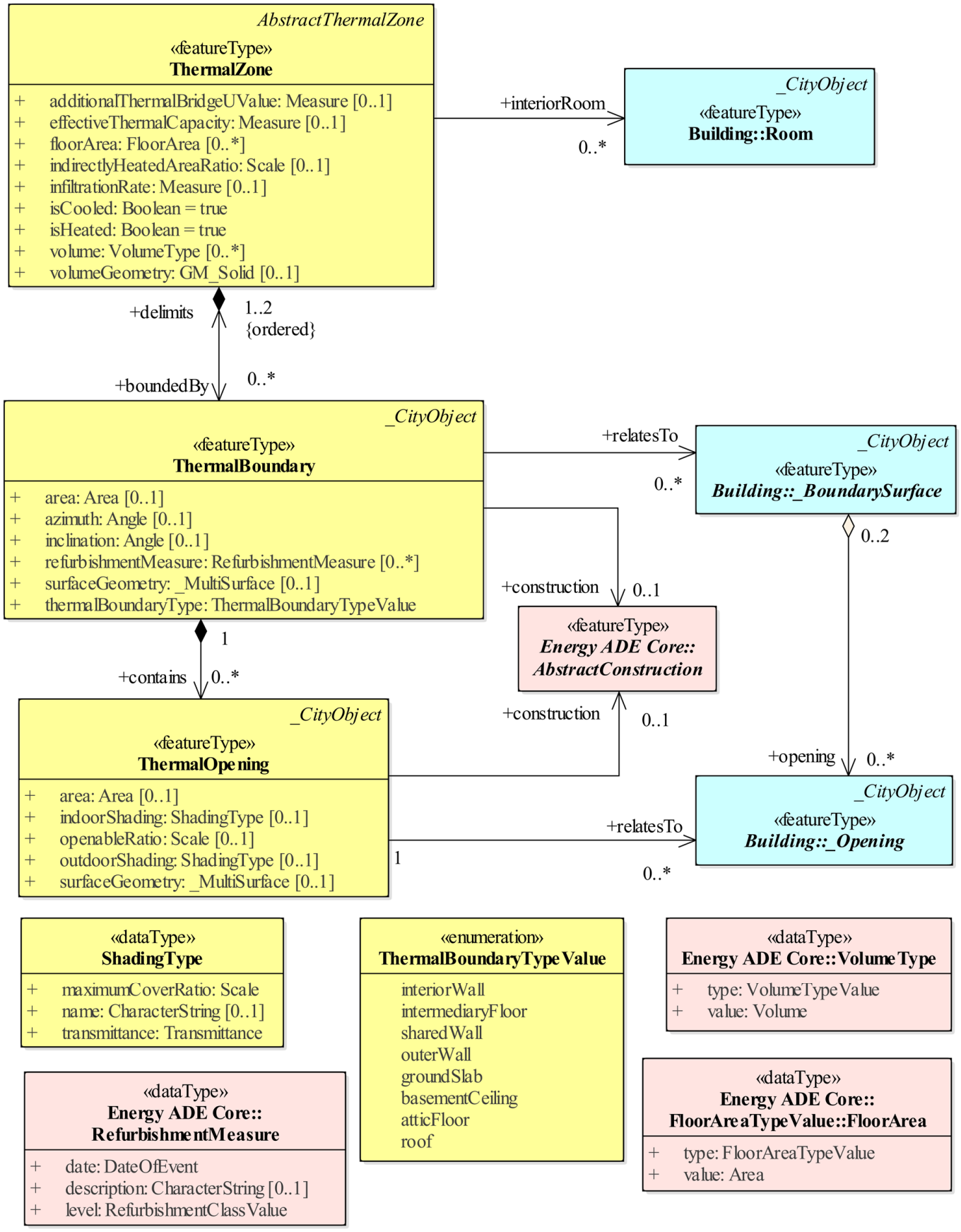

Figure 12. The Building physics module in the Energy ADE for CityGML 2.0.

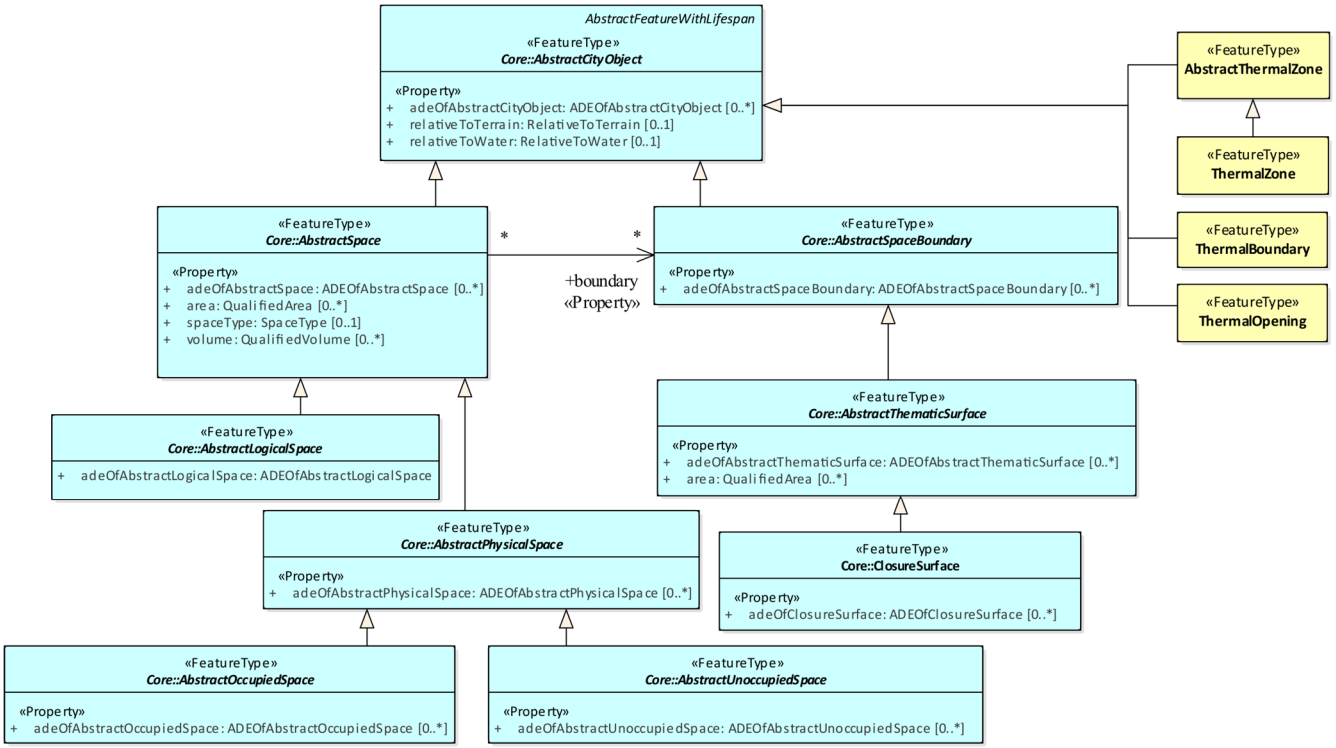

Figure 13. Option to map the Building Physics module classes to CityGML 3.0 by deriving them all from *AbstractCityObject* and, thus, keeping them closer together within the UML class diagram.

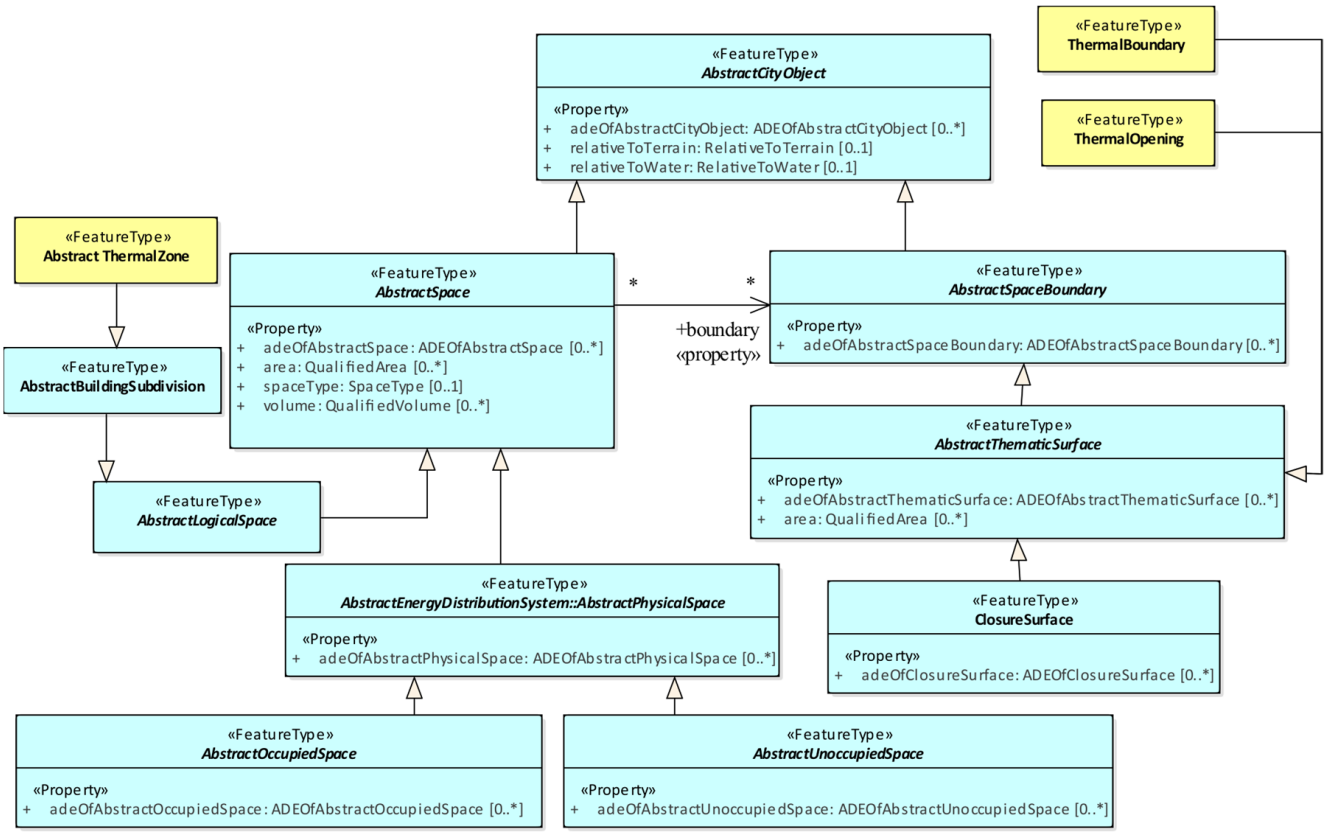

Figure 14. Option to map the Building physics module classes to CityGML 3.0 by their best semantic match.

4.5.1. AbstractThermalZone and ThermalZone

In CityGML 3.0, an *AbstractLogicalSpace* class and its subclasses are defined via thematic considerations and, thus, they fit the intrinsically logical concept of a *ThermalZone*. Although *AbstractLogicalSpace* is a suitable superclass itself, it is relatively generic compared to its more specialised subclasses. Moreover, the mapping principles foresee the integration of ADE classes as deep as possible into the CityGML 3.0 data model to add value. Hence, a closer look at *BuildingUnit* as a potential parent class is taken. A *BuildingUnit* is a "logical subdivision of a Building [. . .] formed according to some homogeneous property" [7]. In the case of the *ThermalZone* class, this homogeneous property relates to the isothermal volume making up a thermal zone. However, having *BuildingUnit* as the parent class for *ThermalZone* results in an interrelation conflict with the Energy ADE class *BuildingUnit* in the Occupant behaviour module. Anticipating some mapping decisions in the Occupant behaviour module, the ADE *BuildingUnit* is merged into the CityGML 3.0 *BuildingUnit* by adding properties via the usual ADE hook mechanism. However, these additional properties could then also be inherited by *ThermalZone*, eventually leading to a logical inconsistency (the details of this reasoning will become more evident when describing the mapping of the Occupant behaviour module in Section 4.7). In order to avoid such logical inconsistencies, *AbstractThermalZone* is subclassed from *AbstractBuildingSubdivision*. Figure 15 illustrates the different mapping options.

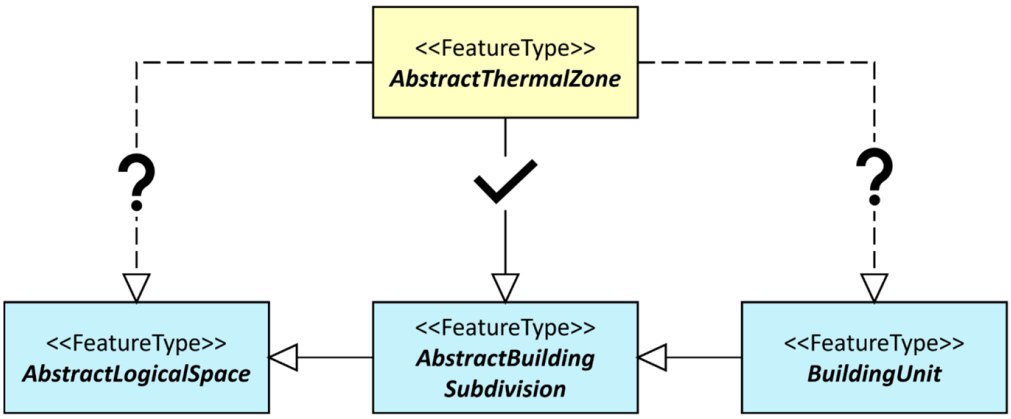

Figure 15. Example of several options for the parent class of *AbstractThermalZone*. Eventually, *AbstractBuildingSubdivision* is chosen.

As seen before, the attributes *floorArea* and *volume* can be replaced through this mapping by *area* and *volume* of *AbstractSpace*. Furthermore, the geometry property *volumeGeometry* of *ThermalZone* can be replaced (i.e., inherited) by the geometry defined in the CityGML 3.0 Core module.

4.5.2. ThermalBoundary and ThermalOpening

In order to utilise the CityGML 3.0 properties for the Energy ADE *ThermalBoundary* and *ThermalOpening* classes, they have to be derived from the class *AbstractThematicSurface* or one of its specialised thematic surface classes. For visual reference, an excerpt of the CityGML 3.0 UML class diagram for thematic surfaces is provided in Figure 16.

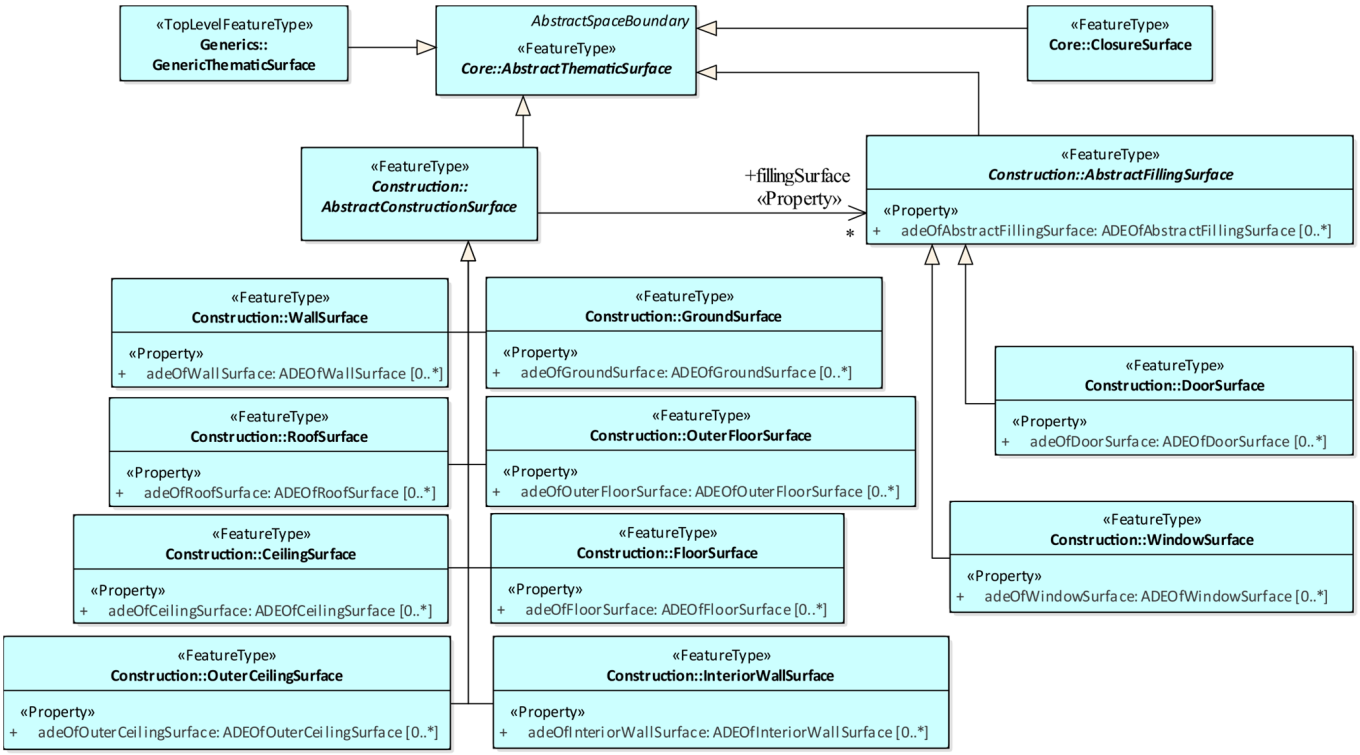

Figure 16. Excerpt of the CityGML 3.0 Construction module showing the different thematic surfaces.

Again, several mapping possibilities exist. One option is to derive the class *ThermalBoundary* from *AbstractConstructionSurface* and the class *ThermalOpening* from *AbstractFillingSurface*. In this case, the relation *fillingSurface* between the CityGML 3.0 parent classes (see Figure 16) could replace the *contained* relationship between *ThermalBoundary* and *ThermalOpening* in the Energy ADE (see Figure 12).

Although the class *ThermalOpening* fits semantically well with *AbstractFillingSurface*, there is a slight mismatch between *ThermalBoundary* and *AbstractConstructionSurface*. The *AbstractConstructionSurface* class is meant to bind CityGML 3.0 *Construction* features (a subclass of *AbstractOccupiedSpace*). However, a *ThermalZone* is a logical space and does not fall under the category of a construction. Hence, *ThermalZone* cannot be bound by a construction surface.

Technically, it is possible to model the Energy ADE classes at different "levels" in the CityGML 3.0 UML diagram. But according to the general mapping principles, similar classes should ideally be derived from the same or comparable parent classes. As such, logical consistency and therefore an easier understanding of the UML diagrams can be ensured. Eventually, a semantically correct mapping, together with the principle of maintaining logical symmetry, outweighs the deeper integration into the CityGML 3.0 UML data model. As a result, *ThermalBoundary* and *ThermalOpening* are both mapped to the more generic CityGML 3.0 class *AbstractThematicSurface*. Regarding attributes, *area* in *ThermalBoundary* and *ThermalOpening* can be replaced by the *area* property of *AbstractThematicSurface*. The surface geometries are also replaced by the corresponding CityGML 3.0 geometries defined in its Core module.

The complete UML diagram of the resulting mapped Building Physics module is shown in Figure 17.

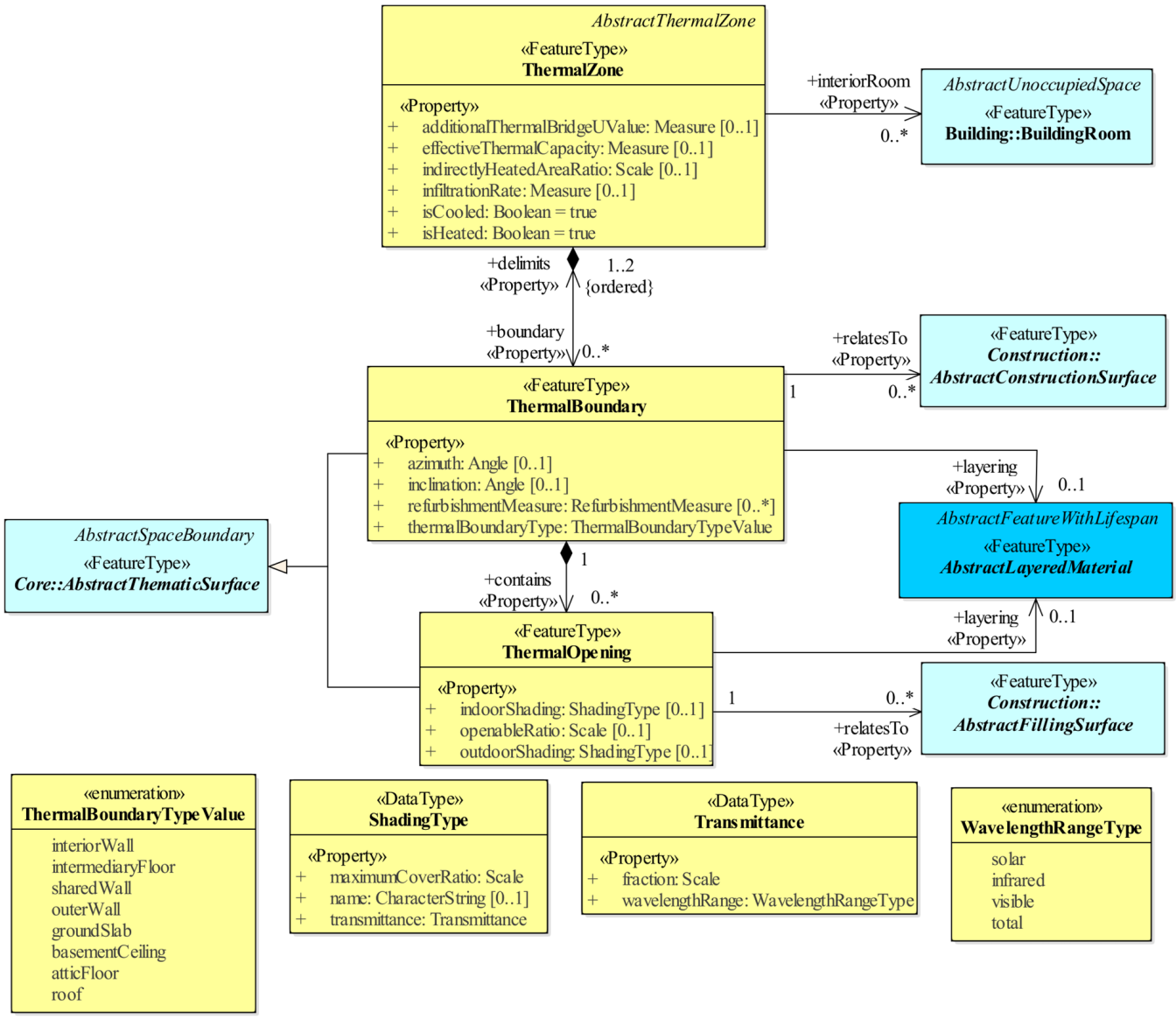

Figure 17. The Building physics module in the Energy ADE for CityGML 3.0.

4.6. The Occupant Behaviour Module in the Energy ADE for CityGML 2.0

In the Energy ADE, the Occupant behaviour module defines classes to model different usage zones and how they are utilised by occupants and facilities such as electrical appliances (see Figure 18). By including schedules, it is possible to represent their behaviour over the day, year, etc. Central to the module is the class *UsageZone*, which defines regions of homogenous usage with regard to their occupants and included facilities. Its properties describe factors affecting the indoor temperature (*heatingSchedule*, *coolingSchedule*, *ventilationSchedule*) and the usage type (*usageZoneType*). Moreover, a *UsageZone* may contain several *BuildingUnit* instances, which specify ownership information. To further specify internal heat gains, *BuildingUnit* and *UsageZone* both have relations to *Occupants* and *Facilities* (*LightingFacilities*, *DHWFacilities*, *ElectricalAppliances*).

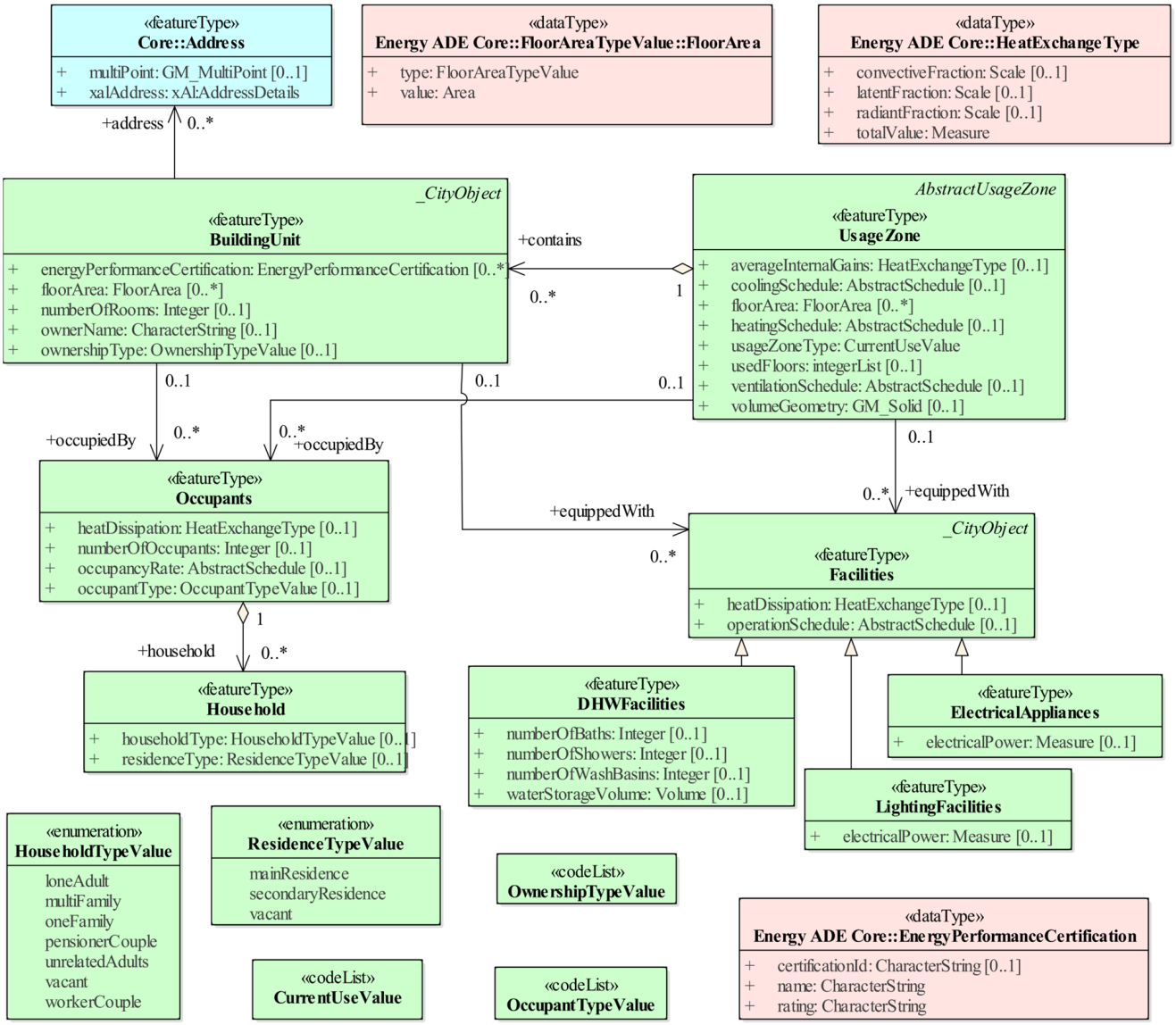

Figure 18. The Occupant behaviour module in the Energy ADE for CityGML 2.0.

Similarly to what was mentioned before for the Building physics module, class *UsageZone* can be optionally associated with a solid geometry. Also, in this case, the *volumeGeometry* property allows only for a single representation and, thus, is decoupled from the usual LOD representation of CityGML.

4.7. Mapping the Occupant Behaviour Module to CityGML 3.0

4.7.1. AbstractUsageZone and UsageZone

Given the previous definition of the CityGML 3.0 *BuildingUnit* class, it qualifies as a fitting parent class for the Energy ADE's *UsageZone*. Since *UsageZone* shows similar traits to *ThermalZone* in the Building physics module, the same issue of potentially inheriting unwanted properties occurs when the class *BuildingUnit* is extended via the ADE hook (see Figure 19). Consequently, *AbstractUsageZone* is also mapped to the next higher generalisation class, *AbstractBuildingSubdivision*. As a result, *BuildingUnit* does not serve as a generalisation class for *AbstractUsageZone* and *AbstractThermalZone* and unwanted properties are not passed on to them. In addition, this solution satisfies the principle of logical symmetry between the two similar classes *ThermalZone* and *UsageZone*.

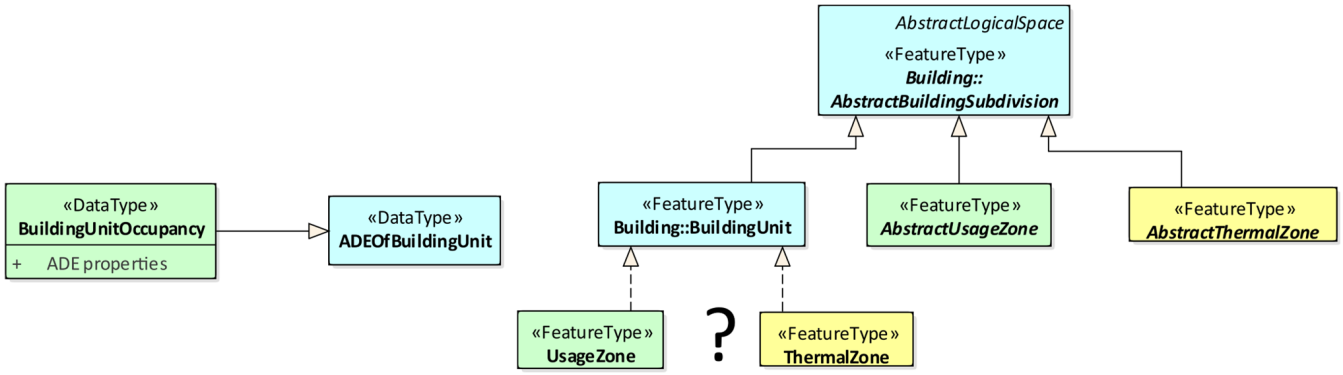

Figure 19. Example of a problematic mapping scenario. When rigidly sticking to the mapping principles, *UsageZone* and *ThermalZone* should both be derived from CityGML 3.0's *BuildingUnit*. As *BuildingUnit* is extended by the ADE properties of class *BuildingUnitOccupancy*, the ADE properties would also be inherited by *UsageZone* and *ThermalZone*, which is not desired. If both classes are derived instead from *AbstractBuildingSubdivision*, the mapping does not adhere to the "integrate as much as possible" principle, however, it solves the aforementioned problem of undesired class inheritance.

4.7.2. BuildingUnit

The concepts of the CityGML 3.0 *BuildingUnit* and of the Energy ADE *BuildingUnit*, which specifies ownership information, match rather well. Because the classes already have the same name and also fit semantically, the CityGML 3.0 class is extended through the ADE hook mechanism to include the additional properties (via the DataType *BuildingUnitOccupancy*). This eventually leads to the mapping shown in Figure 20.

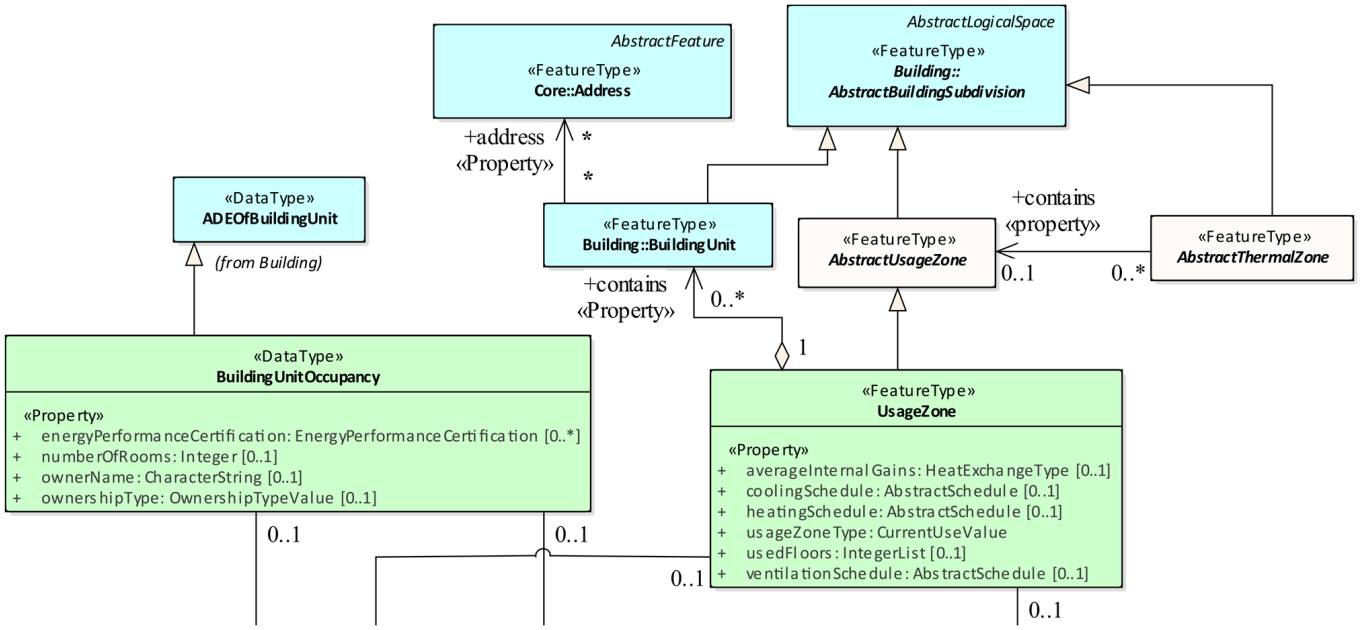

Figure 20. Excerpt of the Occupant behaviour module in the Energy ADE for CityGML 3.0. The full module is depicted in [15].

A positive side-effect of this mapping solution is that *BuildUnit* now inherits a property for a volumetric geometry through its integration with the CityGML 3.0 space and geometry concept. The property *floorArea* is mapped to the corresponding CityGML 3.0 property and the relation to *Address* is also already provided.

4.8. The TimeSeries Classes in the Energy ADE for CityGML 2.0

In the Energy ADE, the time series classes are meant to facilitate the modelling of time-varying attribute values. For this, properties in other modules have the property type *AbstractTimeSeries* (see e.g., property *energyAmount* of class *EnergyDemand* shown in Figure 21).

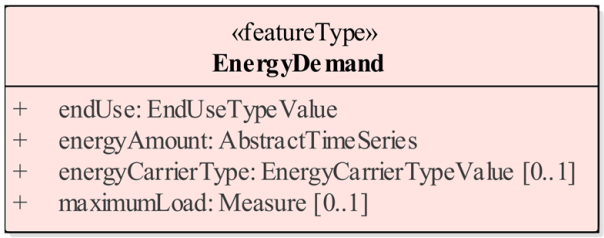

Figure 21. Class *EnergyDemand* in the Energy ADE for CityGML 2.0 with the property *energyAmount* which references a time series through its property type *AbstractTimeSeries*.

Figure 22 shows how the *AbstractTimeSeries* class is further specialised into four subclasses to deal with either regular or irregular time series, possibly stored in-line or in external files. Regular time series have a given time period (*temporalExtent*) and time interval (*timeInterval*) for the measurements. Irregular time series, on the other hand, provide a specific timestamp for every measurement value. Additionally, some metadata can be provided via associated enumeration classes.

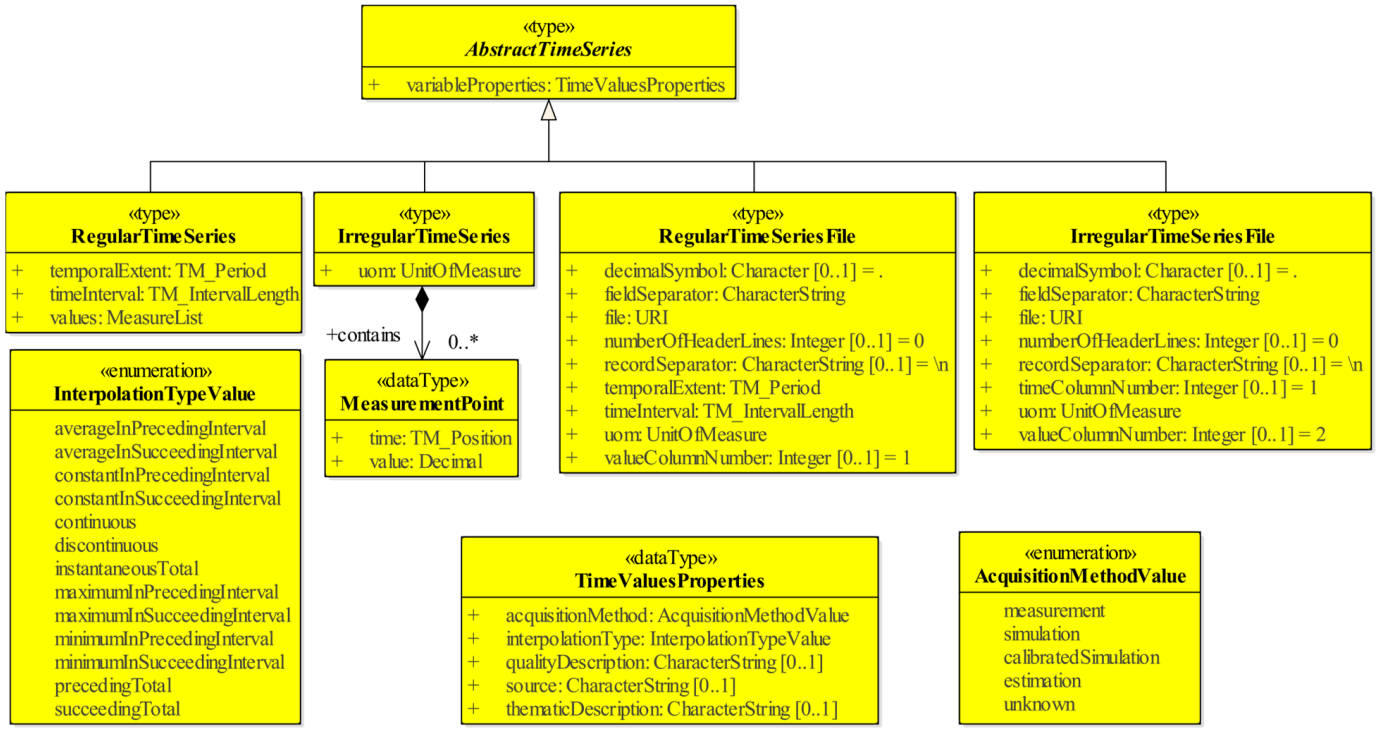

Figure 22. The time series classes in the Energy ADE for CityGML 2.0.

4.9. Mapping the TimeSeries Classes to the Dynamizer Module in CityGML 3.0

One of the major additions to CityGML 3.0 is the ability to model time-dependent attribute values by means of the Dynamizer module. A Dynamizer object can be associated with each property of an *AbstractCityObject* class (and therefore all its subclasses) via the relation to *AbstractDynamizer*. The details on the modelling of such properties are beyond the scope of this article; however, further information can be found in [17].

In the Energy ADE, the classes *EnergyDemand*, *WeatherData* and *EnergyFlow* have the time-dependent properties *energyAmount*, *values*, and *energyAmount*, respectively. However, as these classes are derived from *AbstractFeatureWithLifespan* and *not* from *AbstractCityObject*, they do not inherit the relation to *AbstractDynamizer*. Thus, a new ad hoc relation must be modelled from the respective ADE class to *AbstractDynamizer*. Because the time-varying properties in the Energy ADE are mandatory, they are required to be referenced by a *Dynamizer* instance. This is emphasised through the multiplicity of 1 from the respective ADE class to *AbstractDynamizer*. In addition, a descriptive note states in the UML diagram which of the properties is to be referenced by *Dynamizer*.

With the new modelling technique of time-varying properties, their property types also need to be updated, as they are now expressed as a static value in the respective class. Therefore, as time series consist of values of complex type measure (i.e., value + unit of measure), their type must be set to *Measure*. An example in terms of UML is shown in Figure 23 for the classes *EnergyDemand* (property *energyAmount*) and *WeatherData* (property *values*).

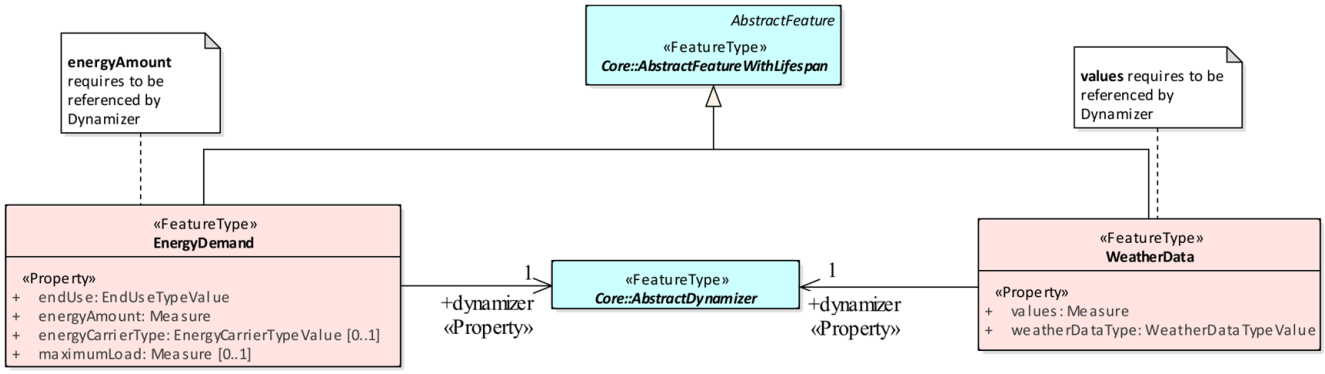

Figure 23. Excerpt of the Energy ADE for CityGML 3.0, showcasing the UML modelling of time-varying properties.

4.9.1. IrregularTimeSeries

The class *IrregularTimeSeries* of the Energy ADE conceptually corresponds to the class *GenericTimeseries* of the Dynamizer module. 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*) and, thus, can be mapped directly. However, the *uom* attribute of the class is mapped to the *uom* attribute of the class *AbstractAtomicTimeseries*.

4.9.2. IrregularTimeSeriesFile

In a similar way, *IrregularTimeSeriesFile* is mapped to *TabulatedFileTimeseries*. Only the property *recordSeparator* cannot be mapped to any of the CityGML 3.0 properties and is therefore added via the ADE hook mechanism through the new data type subclass *TabulatedFileTimeseriesExtension*. In addition to the CityGML 3.0 class *TabulatedFileTimeseries*, another class also handles externally stored time series: *StandardFileTimeseries*. It references files in standardised formats such as the OGC Observations & Measurements Standard or OGC TimeseriesML [7]. Nevertheless, mapping *IrregularTimeSeriesFile* to this class is not suitable as this would require altering the input file or losing properties.

4.9.3. RegularTimeSeries

In the Energy ADE, *RegularTimeSeries* stores an array of time-dependent values together with its total temporal extent, defined as start and end timestamps, and the interval between the timestamp of each value. However, the Dynamizer module of CityGML 3.0 does not offer an equivalent class to the Energy ADE *RegularTimeSeries*. The closest candidate would be *GenericTimeseries*, which however requires that each time-dependent value be stored together with its accompanying timestamp. The additional timestamps

for each value could be computed using the information provided by the original Energy ADE *RegularTimeSeries* data. Choosing the *GenericTimeseries* as target class would, however, lead to a far less compact representation compared to the Energy ADE *RegularTimeSeries*. Thus, opting for this mapping strategy—besides being rather impractical—contradicts the purpose of a compact encoding by the *RegularTimeSeries*.

As a result, an alternative mapping strategy is preferred: A new class *RegularTimeseries* (please note the small s in Timeseries to match the naming style of other Dynamizer classes) is derived from *AbstractAtomicTimeseries*. The goal is to overcome the above-mentioned limitations. Furthermore, the attribute *temporalExtent* is mapped to the properties *firstTimestamp* and *lastTimestamp* of the Dynamizer class *AbstractTimeseries*.

4.9.4. RegularTimeSeriesFile

Also, for the Energy ADE class *RegularTimeSeriesFile* there is no predefined class in the Dynamizer module. The closest option, *TabulatedFileTimeseries*, requires a value for either *timeColumnNo* or *timeColumnName*, meaning that a column containing the timestamps *must* be specified. However, such a column does not exist in a regular time series file. Several options were considered on how to best map the *RegularTimeSeriesFile* to the Dynamizer module. Among them are manually adapting the input file, creating a separate ADE class, or creating a shared *AbstractRegularTimeseries* class for *RegularTimeseries* and *RegularTimeseriesFile*. All of them are discussed in detail in [15].

Eventually, the implemented mapping uses the *TabulatedFileTimeseries* nonetheless, but with a workaround for the OCL constraint. One of the required properties, which indicates the column for the timestamps in the referenced file (*timeColumnName*), asks for a CharacterString data type. When using the *TabulatedFileTimeseries* class for regular time series files, this property can simply be given a NaN (Not a Number) or string value expressing that such a column is not included. Additionally, the ADE property *timeInterval* is added to *TabulatedFileTimeseries* via the ADE hook mechanism. This solution has the advantage of using existing classes rather than creating new ones. Therefore, the UML model remains more compact and avoids modelling repetitive information. Last but not least, it follows the mapping principles of logical symmetry and integrating as much as possible. The resulting final UML class diagram, covering the mapping of the Energy ADE classes for time series, is provided in Figure 24.

4.10. The Schedules Classes in the Energy ADE for CityGML 2.0

Class *AbstractSchedule* and its subclasses, as seen in Figure 25, are part of the Supporting classes module of the Energy ADE. Therefore, they are referenced by the other modules in a similar way as the *AbstractTimeSeries* class. Schedules are used to describe to which extent features or appliances are operated in a certain time period.

The specialisation classes of *AbstractSchedule* are characterised by increasing degrees of freedom regarding how the schedules can be designed. The most general option is *ConstantValueSchedule* which specifies one single value for average usage. Further, *DualValueSchedule* differentiates between idle and operating times. The *DailyPatternSchedule* models change operation times based on the period of the year and the day. Lastly, the *TimeSeriesSchedule* gives complete freedom by modelling the usage through a custom-defined time series.

4.11. Mapping the Schedule Classes to CityGML 3.0

When it comes to mapping the Energy ADE schedules, no directly corresponding concept exists in CityGML 3.0. Thus, they can be mapped in a simpler way than the time series, although some adjustments are still required.

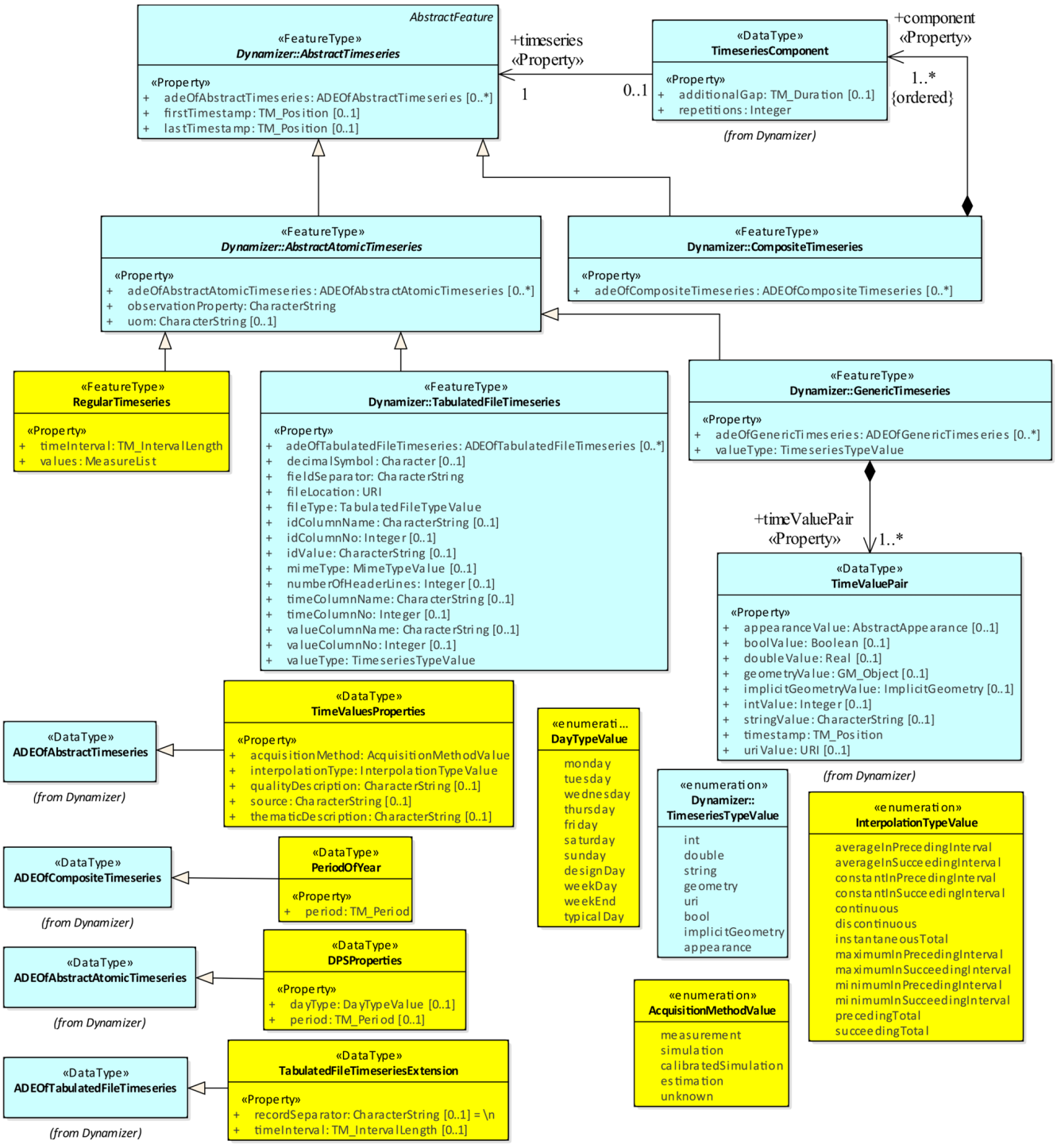

Figure 24. The Dynamizer module of CityGML 3.0 (in cyan) extended with the mapped time series of the Energy ADE (in yellow).

For example, in the Energy ADE, the classes have the stereotype «type». Within CityGML 3.0, this stereotype is not used anymore for application schemas. Nevertheless, to be able to reference the schedules via XLinks, as is very often the case in this context, the new stereotype requires a unique identifier. Because of this reason, «DataType» cannot be used for this purpose. Instead, the classes are given the stereotype «FeatureType».

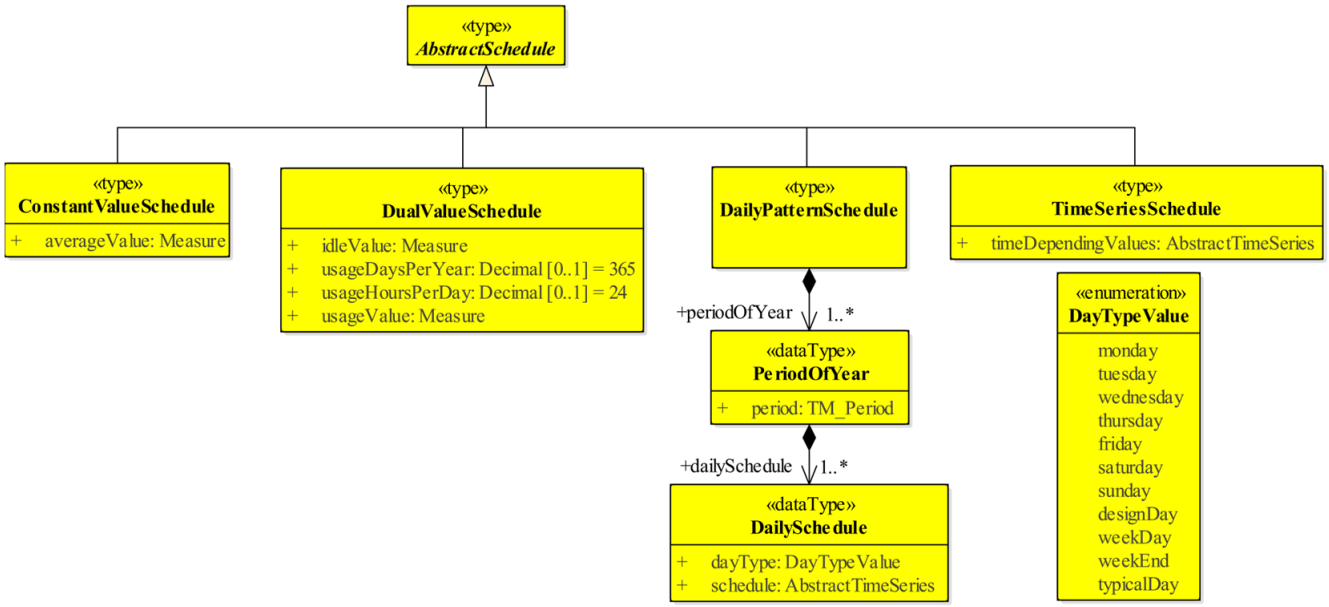

Figure 25. The Schedule classes in the Energy ADE for CityGML 2.0.

As *AbstractSchedule* needs to be linked to one of the existing classes within the CityGML 3.0 model, the parent class *AbstractFeatureWithLifespan* is selected. Choosing instead *AbstractCityObject* as a parent class would not be a conceptually logical solution, because schedules are neither a city object nor do they have a spatial extent. At a higher level, *AbstractFeature* would be a possible choice, as it is more general. Still, in coherence with the *general mapping principles*, this is not the preferred option. *AbstractFeatureWithLifespan* offers instead a deeper integration into the data model and furthermore ensures logical symmetry with *AbstractDynamizer*, which also derives from it. The excerpt from the UML class diagram depicted in Figure 26 shows these relations.

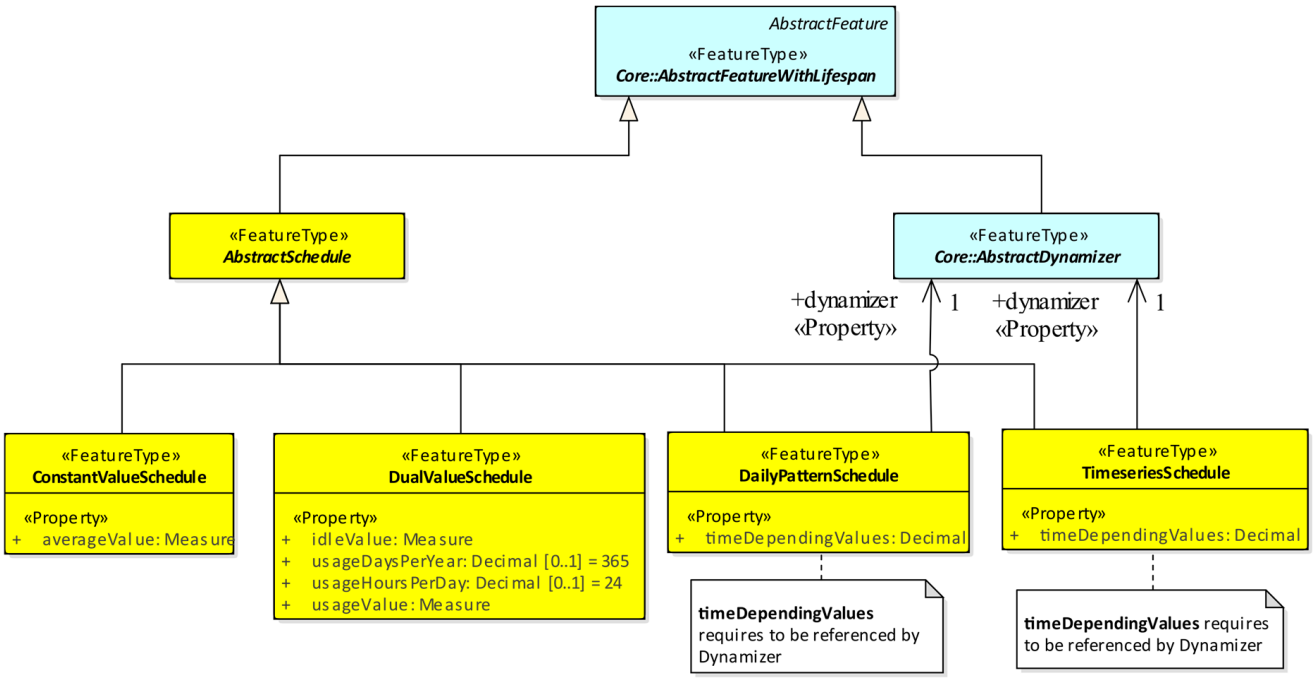

Figure 26. The *Schedule* classes of the Energy ADE for CityGML 3.0.

The properties of the mapped Energy ADE classes that are described via schedules have now the property type *AbstractSchedule*. This in-line representation (in the original Energy ADE) is de facto the same as the relation by reference from a feature type to *AbstractSchedule* (see Figure 27) in the mapped version for CityGML 3.0.

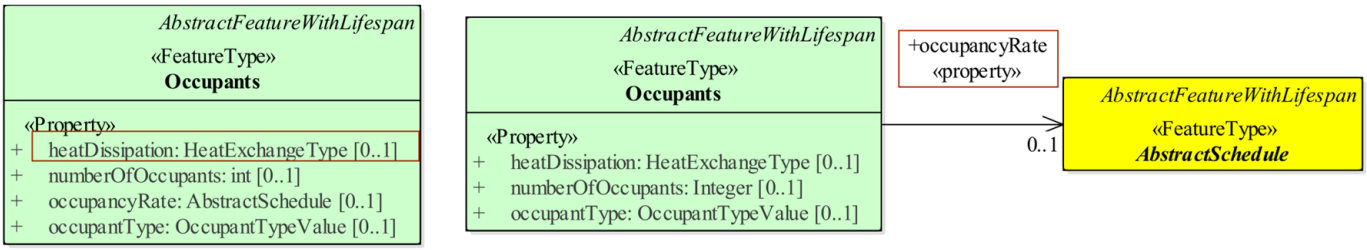

Figure 27. The property *occupancyRate* represented in-line and highlighted in red (left) equals the by-reference representation (right).

4.11.1. ConstantValueSchedule and DualValueSchedule

The classes are changed to «FeatureType» and are subclassed from *AbstractSchedule*. None of their properties can be mapped to CityGML 3.0, therefore, the overall structure remains nearly identical to the original Energy ADE.

4.11.2. TimeSeriesSchedule

The only property *timeDependingValues* specifies, as a ratio, how much something is used over a given time and, as such, does not need a unit of measure. The time-dependent values can be modelled through a connection to *AbstractDynamizer*. Here, the user is free to choose which class in the Dynamizer module best describes the intended time series.

4.11.3. DailyPatternSchedule

Two options for mapping the class *DailyPatternSchedule* were considered. In the first one, the original Energy ADE structure (i.e., as compositions of *PeriodOfYear* and *DailySchedule*) is simply recreated for CityGML 3.0. Alternatively, the *CompositeTimeseries* and *TimeseriesComponent* in the Dynamizer module are used to re-model the nested structure of the Energy ADE class *DailyPatternSchedule*. This second mapping choice is made possible in CityGML 3.0 because the class *CompositeTimeseries* can contain multiple instances of the class *TimeseriesComponent*, which are themselves associated with any of the available time series derived from the class *AbstractTimeseries* (see Figure 28).

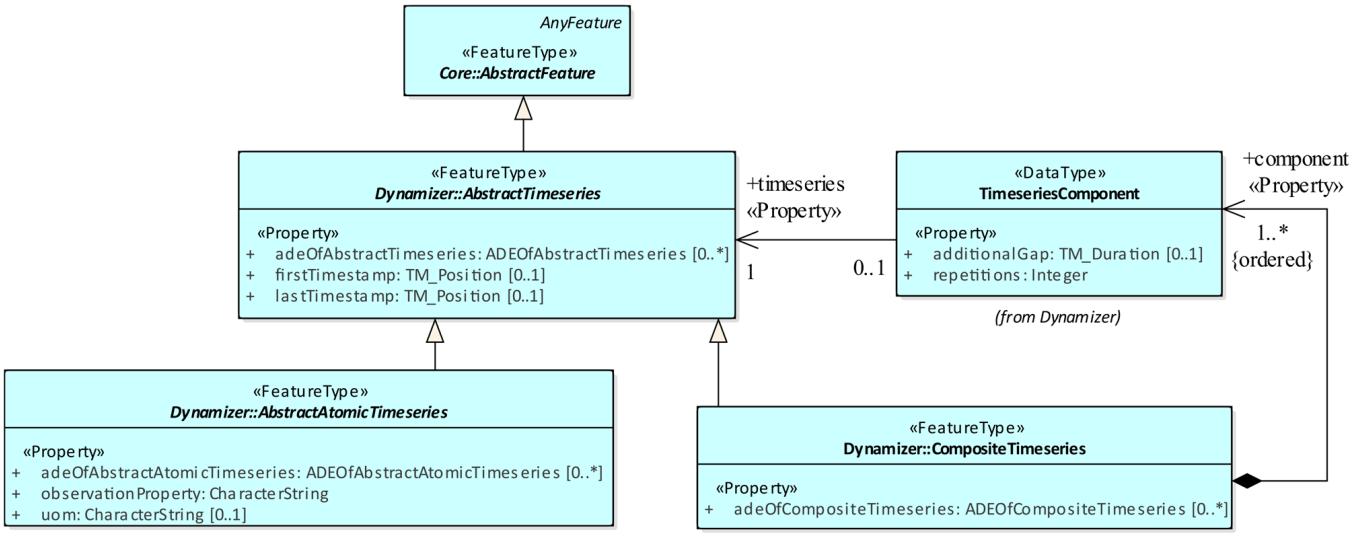

Figure 28. Excerpt of the CityGML 3.0 Dynamizer module.

To pursue this second modelling approach, the Energy ADE class *DailySchedule* is mapped to the data type *TimeseriesComponent*. Yet, its property *dayType* cannot be included in *TimeseriesComponent*, because the ADE hook mechanism does not apply to data types. As a workaround *dayType* is added to *AbstractAtomicTimeseries* through an ADE hook instead because every *TimeseriesComponent* is eventually described by the other time series. Furthermore, the *period* property of the class *PeriodOfYear* has to be mapped to two different classes within the Dynamizer module due to the flexibility the nested structure of *DailyPatternSchedule* gives. If a *DailyPatternSchedule* has only one time period, the property can be added to *CompositeTimeseries* with the ADE hook mechanism. If a *DailyPatternSchedule* has multiple time periods (*PeriodOfYear*), the *period* property is directly attached to the time series. This is realised through a hook to the class *AbstractAtomicTimeseries*. The new properties added to the Dynamizer module for the *DailyPatternSchedule* are summarised in Figure 29.

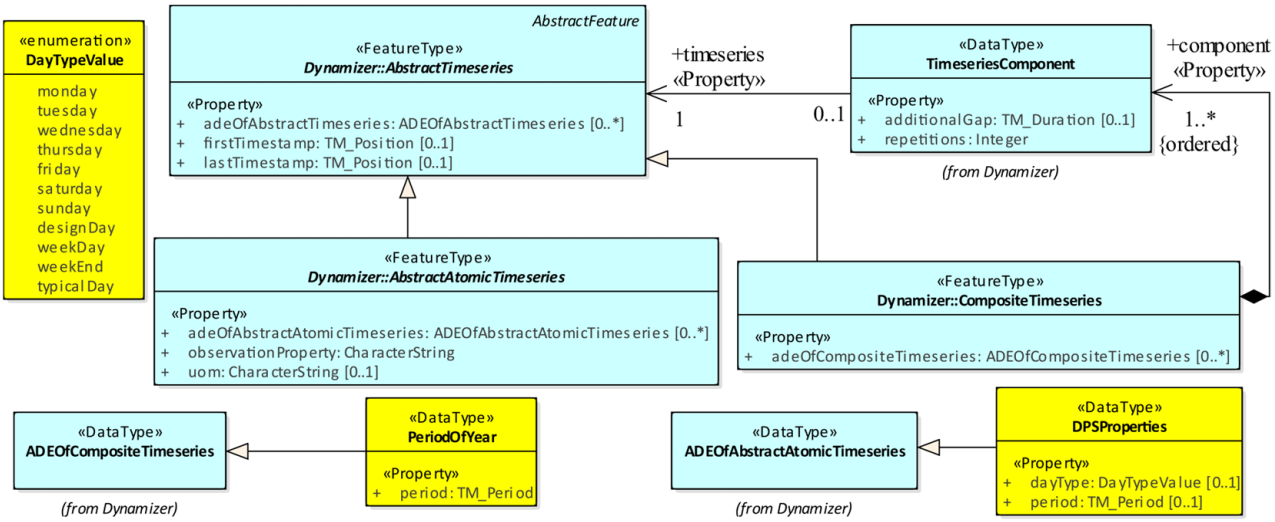

Figure 29. Excerpt of the CityGML 3.0 Dynamizer module (in cyan) with the added properties (in yellow) to map the DailyPatternSchedule.

With this mapping solution, even though it is rather complex, the class *DailyPatternSchedule* for CityGML 3.0 requires only the property *timeDependingValues* and a connection to *AbstractDynamizer*. As such, it makes use of the concepts already available in CityGML 3.0 and adheres to the *general mapping principles*.

5. Further Steps

Following the UML-based mapping, the data model is first derived as an XSD schema file in order to create and validate respective data. The applicability of the final mapping is then tested with a sample dataset which is converted to CityGML 3.0 + "new" Energy ADE.

5.1. XSD Schema Derivation

For the XML-based encoding of CityGML (and any associated ADEs), XML Schema Definition (XSD) files are required. They encode, in a machine-readable way, the data model and its constraints therefore defining how data can be written and automatically validated.

The required XSD schema for the Energy ADE for CityGML 3.0 is automatically derived from the UML class diagrams by means of the software tool ShapeChange v. 2.11. In order to do so, ShapeChange requires a custom configuration file which specifies, for example, the UML diagrams to process, the target encoding and the output directory.

To simplify the task, an already existing configuration file from the Utility Network ADE [14] has been adapted to match the requirements of the Energy ADE. The resulting XSD file can be found on GitHub [18]. The generated XSD file was carefully checked also manually to ensure the correctness of the classes and properties.

5.2. Test Data Creation and Conversion

To test the validity of the "new" Energy ADE for CityGML 3.0, a test dataset with CityGML 2.0 and Energy ADE 1.0 data was first created and then successively converted to a dataset with the "new" Energy ADE for CityGML 3.0. The test dataset contains every Energy ADE class and property at least once in order to verify that the data conversion is carried out correctly and without any loss of data. Both of these steps are implemented in an FME Workbench and are briefly described in the following. Both the FME workbench and the test dataset can be retrieved from the GitHub repository, too.

The test dataset builds upon an artificial CityGML 2.0 city model with 12 buildings (as seen in Figure 30) that are already enriched with some Energy ADE properties. They are modelled in LOD2 through their boundary surfaces *WallSurface*, *RoofSurface* and *GroundSurface*. In addition, the buildings have geometries via the *referencePoint* and *lod0FootPrint* properties. Every building has one *ThermalZone* and one *UsageZone* with geometries following the CityGML boundary surfaces. Moreover, *ThermalBoundary* and *ThermalOpening* are defined by *Constructions* through *Layer*, *LayerComponent* and *Material*. Additionally, each building has a set of *Households*, *Occupants* and *Facilities* as well as an *occupancyRate* schedule and an *EnergyDemand* time series. Lastly, the test data have one *WeatherStation* containing temperature and humidity information.

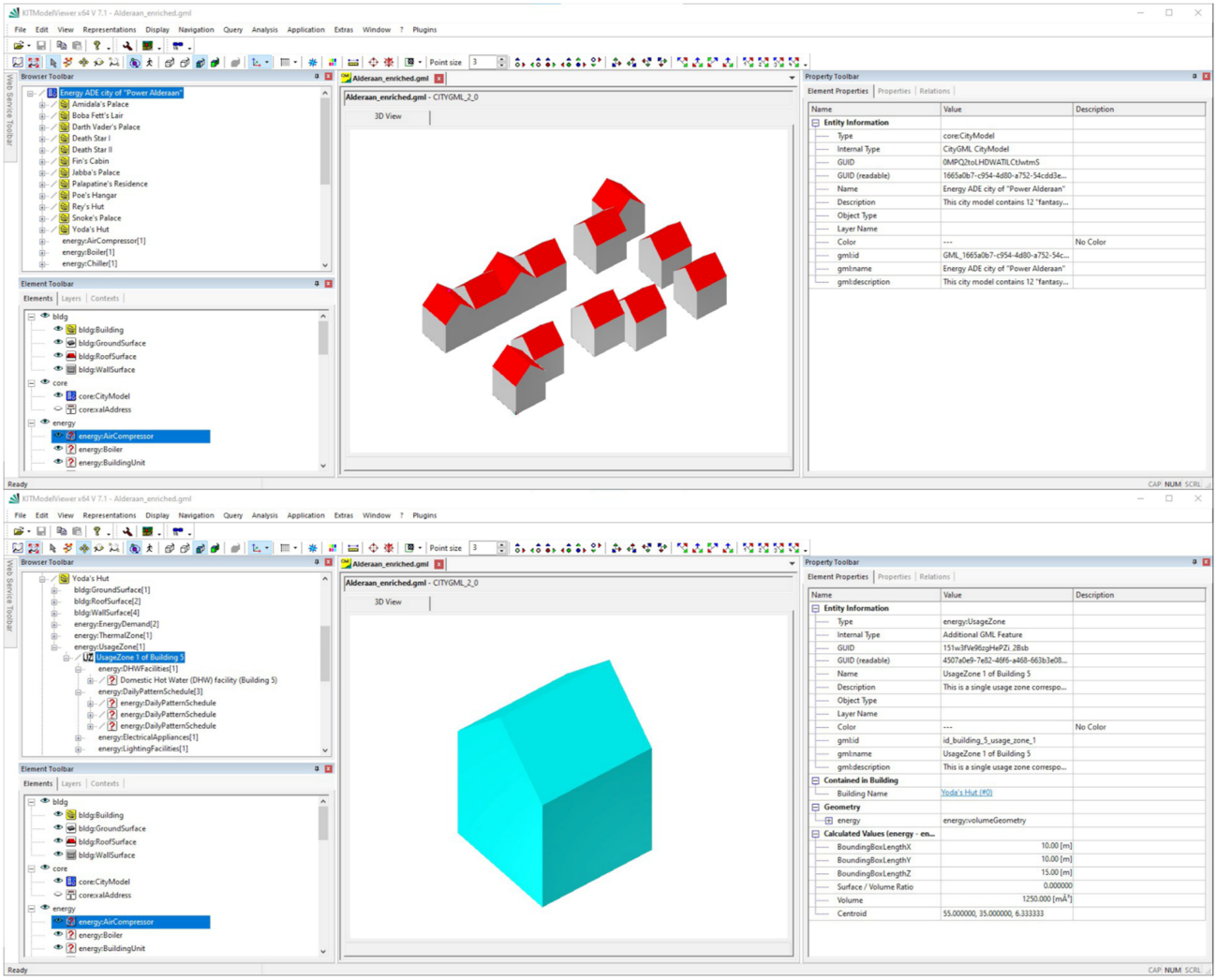

Figure 30. Visualisation of the test dataset in the FZK ModelViewer. Upper picture: the CityModel with its LOD2 buildings and properties. Lower picture: the *UsageZone* of Building "Yoda's Hut" with its properties and own geometry.

All classes and properties that are not already present in the original test dataset are added through an FME Workspace. This mainly includes data covering the whole Energy systems module, some additional individual properties and feature types, as well as examples of *RegularTimeSeries* and *DailyPatternSchedule*. Eventually, the created dataset covers all important aspects of the Energy ADE and serves as input for the conversion to Energy ADE for CityGML 3.0.

For the conversion, an FME template that transforms the Building module to CityGML 3.0 is used as a starting point [16]. The data are imported with a CityGML Reader and exported with a GML Writer which is provided with the XSD schema files (for both CityGML 3.0 and the "new" Energy ADE). The reason for doing so is that FME did not support CityGML 3.0 natively when the mapping was carried out (beginning of 2023).

In this context, only the main overarching concepts of the conversion are presented. A more detailed explanation is provided in [15] and in the GitHub repository [18]. Figure 31 provides a schematic overview of the whole conversion process in FME. Large parts are dedicated to renaming attributes according to their altered FME encoding. Moreover, ADE geometries are transferred to the corresponding CityGML 3.0 ones wherever possible. Furthermore, *Schedule*, *TimeSeries* and *WeatherData* objects have now their own FME Writer due to their stereotype being changed to «FeatureType» in the Energy ADE for CityGML 3.0. Thus, the associated information is separated and further handled to connect them to their new corresponding FME Writer. Finally, the conversion handles individual changes of mapped properties and property values.

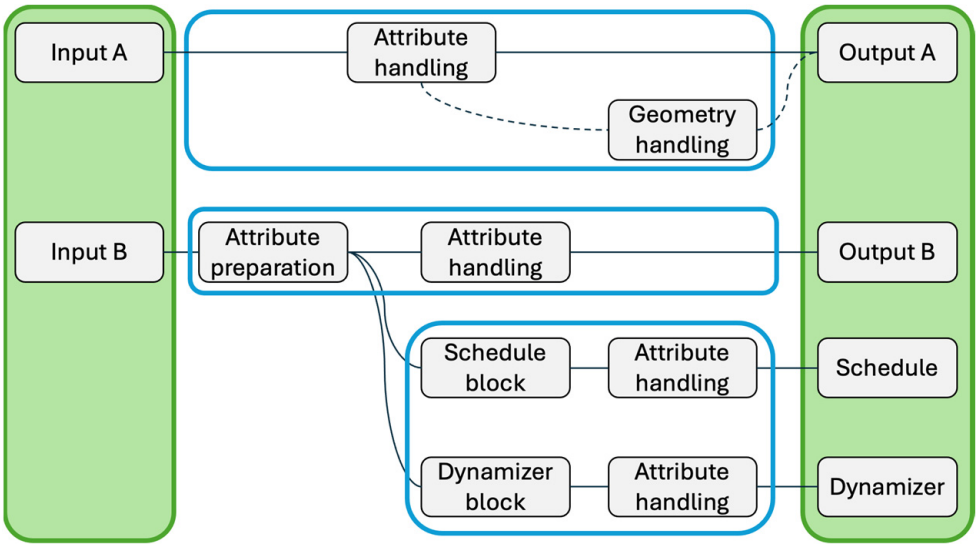

Figure 31. Schematic overview of the data conversion from Energy ADE for CityGML 2.0 to Energy ADE for CityGML 3.0 in FME. The "Input A" block stands for input class A, e.g., *ThermalZone*, and the "Output A" block for the respective output after the conversion. Unlike in CityGML 2.0, the *Schedule* and *Dynamizer* now have their own classes in the Energy ADE for CityGML 3.0, which is why they also have their own blocks to write the final output.

6. Results and Discussion

In this section, the results are presented, and the implemented mapping strategy, its implications, and the lessons learnt are discussed. The reflection will cover mainly the chosen level of integration between the Energy ADE and the CityGML 3.0 data model, the resulting geometry representation and how the gained insights can be beneficial for the development (or conversion) of other ADEs in the context of CityGML 3.0.

Regarding the mapping results, an excerpt is contained in Table 1. It contains three classes, indicating how much they have changed during the mapping process, as well as some relevant details. The classes shown in Table 1 are chosen to provide three representative examples. For space and readability reasons, the table containing all classes is presented in Appendix A.

Table 1. Selected classes representing the results of the mapping. The "Status" column refers to the degree of change through the mapping, while the "Details" column gives condensed information about the mapping.

Energy ADE 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 into space and geometry concept, property *yearOfManufacture* replaced by CityGML 3.0 property
Time Series	*AbstractTimeSeries*	Obsolete	Property *variableProperties* is mapped to *AbstractTimeseries* with the ADE hook
	IrregularTimeSeries/GenericTimeseries	Obsolete	Replaced by *GenericTimeseries* in the Dynamizer module
	RegularTimeSeriesFile, *IrregularTimeSeriesFile/TabulatedFileTimeseries*	Obsolete, Adapted	Both classes largely replaced by *TabulatedFileTimeseries* in the Dynamizer module, addition of properties *recordSeparator* and *timeInterval* with the ADE hook

In the table, the status "Mostly taken over" means that only some minor changes were necessary to fit CityGML 3.0. "Adapted" refers to some major adjustments and "Obsolete" tells that the Energy ADE class was completely replaced by a CityGML 3.0 one. For example, the ADE properties for *_AbstractBuilding* were mapped to native CityGML 3.0 properties wherever possible and were furthermore adapted according to the restructured ADE hook mechanism. These are not structural changes, which is why this mapping is categorised as "Mostly taken over". On the other hand, the class *AbstractEnergySystem* was mapped to a new generalisation class which integrates it into the new space and geometry concept. In the Energy ADE for CityGML 2.0, the class could not be represented geometrically. These changes alter the structure of the Energy ADE class, which is thus assigned the status "Adapted".

6.1. Level of Integration

The resulting data model of the Energy ADE for CityGML 3.0, together with the developed XSD file and FME workbench, prove that a data conversion can be successfully carried out without any data losses. The mapping procedure has followed the guidelines listed in Section 4.1, which allow for a uniform mapping on a logical and conceptual level. However, sometimes there is more than one possible solution to perform the mapping. Thus, for the sake of completeness, two possible alternative mapping strategies are briefly outlined in the following subsections, although they were ultimately not implemented and only the mapping strategy presented in Section 4 was used to obtain the XSD file and the FME workbench. We have called them "minimum" and "middle ground" mapping strategies.

6.1.1. Minimum Mapping

The so-called "minimum mapping" approach could be seen as a sort of brute-force mapping, in which only strictly necessary "technical" adjustments would be made for the Energy ADE to work with CityGML 3.0. In other words, all Energy ADE classes would be derived directly from CityGML 3.0 *AbstractCityObject* or *AbstractFeature*, without any further reasoning on exploiting the new classes and concepts of CityGML 3.0.

Some of the strictly necessary "technical" adjustments would be required due to the revised ADE hook mechanism of CityGML 3.0. Moreover, the generalisation class names would need to be updated according to the new standard (e.g., from *_CityObject* to *AbstractCityObject*). At last, the stereotype «type» in *ServiceLife*, *WeatherData*, the time series and schedules classes would need to be adapted to a viable alternative. It remains open to further investigation how time series could be dealt with in this scenario. The closest option to the original Energy ADE would be to change the classes to the stereotype «FeatureType» and use *AbstractFeature* as the parent class for *AbstractTimeSeries*. This solution, not further followed in our mapping of time series, however, was used to map the Energy ADE schedules.

The result of a "minimum mapping" approach would lead to a resulting data model that is "closer" to the original Energy ADE for CityGML 2.0. On the one hand, it would offer a less complex solution than the proposed one by keeping the classes closer together and deriving them all from CityGML 3.0 classes that are rather high up in the hierarchy (*AbstractFeature*, *AbstractCityObject*). On the other hand, it would not take into account many of the changes in CityGML 3.0. None of the classes would be derived from classes of a lower hierarchy level than *AbstractCityObject* and would thus not utilise the newly introduced space and geometry concept. Therefore, the geometries would need to be explicitly defined, none of the properties could be replaced by CityGML 3.0 ones, and the ADE classes would furthermore not benefit from any of the additionally provided semantics. As a result, a lot of redundant information would be created through this mapping approach, which is not the purpose of an extension of the given data model.

6.1.2. Middle Ground

A second possible alternative could consist of a compromise between the "minimum mapping" approach and its opposite one, i.e., the implemented "integrate as much as possible" approach. Here, the ADE classes could be integrated into the CityGML 3.0 space and geometry concept where the semantic relation is evident. However, only abstract space classes would be considered. *AbstractThermalZone* would then for instance either be subclassed from *AbstractSpace* or *AbstractLogicalSpace*. Furthermore, Energy ADE classes without geometries would remain subclassed from *AbstractCityObject* (i.e., *Facilities*). For the remaining classes, it would remain open to discussion whether to derive them from *AbstractFeature* or *AbstractFeatureWithLifespan*.

This middle-ground solution would utilise the CityGML 3.0 geometries where applicable, while not giving new ones to ADE classes that did not have them before. Additionally, it would also provide some additional contextual information. Thus, this strategy would benefit from some of the updates in CityGML 3.0, and at the same time, keep the Energy ADE closer to the original one.

Both herewith discussed alternative mapping approaches could probably be implemented, also without any loss of information. The only major difference would consist in the different levels of integration with CityGML 3.0 and, therefore, how much additional context is provided. Yet, as opposed to these two mapping options, the actually implemented one accounts for all changes and new features in CityGML 3.0. It adheres to the strategy proposed in the CityGML 3.0 Conceptual Model Standard, i.e., to derive the classes according to their best semantic fit. Moreover, it also complies with the CityGML 3.0 developers' ideal that as little as possible be derived from *AbstractCityObject* itself.

6.2. Geometry Representation

Several Energy ADE classes which were formerly derived from *_CityObject* are now subclassed from CityGML 3.0 classes further down in the hierarchy. This tighter integration with the space and geometry concepts has several advantages.

First, the existing CityGML 3.0 geometries are now reused instead of explicitly defining them in the ADE classes themselves. Through this, a multi-geometry representation in different LODs of the Energy ADE classes is now possible for Energy ADE classes derived from *AbstractCityObject* or its subclasses. In the Energy ADE for CityGML 2.0, only one geometry representation is foreseen per class. Since this restriction does not apply anymore in the case of the Energy ADE for CityGML 3.0, guidelines should be provided on how to best apply these new modelling possibilities. For instance, it is now possible to model a *ThermalZone* in LOD2 or in LOD3. But how and when? We believe that, in general, a good starting point is the common LOD notion as defined in CityGML 3.0, that could be applied and adapted to the use case.

Another result of the deeper integration into the CityGML 3.0 data model is that some Energy ADE classes can now be represented through geometries as opposed to before. This is the case for all classes which formerly derived from *_CityObject* such as *Facilities* or subclasses of *AbstractEnergySystem*. However, their geometric representation remains optional, so that no additional conditions are required.

Nevertheless, some additional specific rules for the geometry representation of individual classes should be defined, such as a maximum LOD or mandatory relations to other classes and their geometries (i.e., geometry representation of *Facilities* only through *lod0Point*). However, these particular definitions go beyond the scope of this research.

Seemingly, this contradicts the aim of this work to map the Energy ADE without any changes in its content and functionalities. While this is true to some extent, the pros of the overall logic and consistent mapping outweigh the resulting cons. 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 functionalities of the Energy ADE.

6.3. Considerations beyond Mapping

The goal of this research was to map the Energy ADE to CityGML 3.0 without changing its content or functionalities. However, the thorough examination of both those data models resulted in insights/possibilities on changes beyond mere mapping. While these options were not implemented, they are nonetheless briefly presented at this point. For a more detailed explanation, the reader is invited to refer to [15].

First, the mandatory relation to the Dynamizer for the Energy ADE classes with time-dependent properties could be instead modelled as optional. This way, such properties could also be represented in a simplified way by a single static value. Second, the ADE class *RegularTimeseries*, which is added to the Dynamizer module, only accepts numeric values for its property *values*. Yet, the similar CityGML 3.0 class *GenericTimeseries* also allows other data types for the values such as Booleans or strings. Allowing them in *RegularTimeseries* as well would make the class more flexible and also more coherent to CityGML 3.0.

These examples show that not all changes of CityGML 3.0 can be used with the applied mapping approach. One reason for this is that the original Energy ADE was developed for CityGML 2.0. A newly modelled Energy ADE specifically for CityGML 3.0 would thus most likely lead to a different result.

On this note, we would also like to suggest the incorporation of class *RegularTimeseries* into future releases of CityGML 3.0. The modelling solution that we propose represents a more space-efficient encoding compared to the current *GenericTimeseries*.

7. Conclusions

The release of CityGML 3.0 comes with many changes, which, on the one hand, imply that Application Domain Extensions developed for its previous version cannot function anymore with the latest version unless some adjustments are made. On the other hand, CityGML 3.0 opens up new opportunities for ADEs to make use of its extended functionalities such as the centrally defined space and geometry concepts, newly introduced classes and properties, as well as the possibility to model time-dependent attribute values using the Dynamizer.

This article has investigated the possibilities of how these changes affect ADEs using the example of the Energy ADE, and how an ADE can be mapped to CityGML 3.0 without reducing its modelling capabilities. The Energy ADE was chosen as it is one of the most complex (and best documented) ADEs currently available for CityGML 2.0 and it covers different data modelling strategies when it comes to extending CityGML, as well as different simple and complex data types, including codelists and enumerations.

The mapping of the Energy ADE from CityGML 2.0 to 3.0 was carried out following a model-driven approach, as it is the suggested approach for CityGML 2.0 ADEs and upon which CityGML 3.0 was developed, too. To test the validity of the mapping and the actual data transformation, a CityGML 2.0 + Energy ADE sample dataset was created and successfully converted to CityGML 3.0 + "new" Energy ADE in FME. The resulting FME workbench, as well as the generated XSD schema file for the Energy ADE, are publicly available via a GitHub repository.

The results show that a mapping of the Energy ADE to CityGML 3.0 is indeed possible. When performing the mapping, the "integrate as much as possible" approach was chosen and implemented, although other alternative approaches (also briefly mentioned in this article) could have been adopted. As a result, the "new" Energy ADE has become more compact through the mapping of attributes and the replacement of all geometries by means of the centrally defined geometry concept in CityGML 3.0. Furthermore, the ADE classes are now semantically richer due to the new space concept and their mapping to more specialised classes within the CityGML 3.0 data model. Some time series classes could be fully replaced by CityGML 3.0 classes in the Dynamizer module.

Of course, only more testing and further implementations will show the overall applicability of the developed mapping approach, for the Energy ADE in this specific case, but also for other existing ADEs to be ported to CityGML 3.0. Nevertheless, given the scarcity of existing publications and documentation on this specific topic, together with the limited number of available examples, we believe that our experience may contribute to narrowing the knowledge gap and serve as an example for other ADEs to follow.

Author Contributions: Conceptualisation, Carolin Bachert, Camilo León-Sánchez, Tatjana Kutzner and Giorgio Agugiaro; methodology, Carolin Bachert, Camilo León-Sánchez, Tatjana Kutzner and Giorgio Agugiaro; software, Carolin Bachert and Tatjana Kutzner; data curation, Carolin Bachert and Giorgio Agugiaro; writing—original draft preparation, Carolin Bachert and Giorgio Agugiaro; writing—review & editing, Carolin Bachert, Camilo León-Sánchez, Tatjana Kutzner and Giorgio Agugiaro; visualisation, Carolin Bachert and Camilo León-Sánchez; supervision, Camilo León-Sánchez, Tatjana Kutzner and Giorgio Agugiaro. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All materials created in this research are available in the GitHub repository [18]. The description of the contents is available in the readme file.

Conflicts of Interest: Author Carolin Bachert was employed by the company con terra GmbH. The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A

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 into space and geometry concept, property *yearOfManufacture* replaced by CityGML 3.0 property
	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 properties
	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 semantic mismatch with CityGML 3.0 concept 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 properties
	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 property
	Occupants, *Household*	Mostly taken over	New generalisation class: *AbstractFeatureWithLifespan*
	Facilities, *DHWFacilities*, *LightingFacilities*, *ElectricalAppliances*	Adapted	New generalisation class: *AbstractOccupiedSpace*, incorporation into space and geometry concept

Module	Class	Status	Details
Energy Systems	*AbstractEnergy ConversionSystem*, *Boiler*, *ElectricalResistance*, *CombinedHeatPower*, *MechanicalVentilation*, *AirCompressor*, *Chiller*, *GenericConversion System*, *HeatPump*, *HeatExchanger*, *AbstractSolarEnergy System*, *Photovoltaic System*, *SolarThermal System*, *Photovoltaic ThermalSystem*	Mostly taken over	Incorporation into space and geometry concept, generalisation class derives from *AbstractOccupiedSpace*
	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*
Support classes: 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 the Dynamizer module, addition of properties *recordSeparator* and *timeInterval* with the ADE hook
Support classes: 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 time series are now covered through *CompositeTimeseries* in the Dynamizer module
	TimeSeriesSchedule/*TimeseriesSchedule*	Adapted	Changed to stereotype «FeatureType», relation to *AbstractDynamizer*
Support classes: other	*WeatherStation*	Adapted	New generalisation class: *AbstractPhysicalSpace*. Incorporation into space and geometry concept

References

1. Nageler, P.; Koch, A.; Mauthner, F.; Leusbrock, I.; Mach, T.; Hochenauer, C.; Heimrath, R. Comparison of dynamic urban building energy models (UBEM): Sigmoid energy signature and physical modelling approach. *Energy Build.* **2018**, *179*, 333–343. [CrossRef]
2. Horak, D.; Hainoun, A.; Neugebauer, G.; Stoeglehner, G. A review of spatio-temporal urban energy system modeling for urban decarbonization strategy formulation. *Renew. Sustain. Energy Rev.* **2022**, *162*, 112426. [CrossRef]
3. Corrado, V.; Fabrizio, E. Steady-State and Dynamic Codes, Critical Review, Advantages and Disadvantages, Accuracy, and Reliability. In *Handbook of Energy Efficiency in Buildings*; Elsevier: Amsterdam, The Netherlands, 2019; pp. 263–294. [CrossRef]
4. Ali, U.; Shamsi, M.H.; Hoare, C.; Mangina, E.; O'Donnell, J. Review of urban building energy modeling (UBEM) approaches, methods and tools using qualitative and quantitative analysis. *Energy Build.* **2021**, *246*, 111073. [CrossRef]
5. Gröger, G.; Plümer, L. CityGML—Interoperable semantic 3D city models. *ISPRS J. Photogramm. Remote Sens.* **2012**, *71*, 12–33. [CrossRef]
6. Agugiaro, G.; Benner, J.; Cipriano, P.; Nouvel, R. The Energy Application Domain Extension for CityGML: Enhancing interoperability for urban energy simulations. *Open Geospat. Data Softw. Stand.* **2018**, *3*, 2. [CrossRef]
7. Kolbe, T.H.; Kutzner, T.; Smyth, C.S.; Nagel, C.; Roensdorf, C.; Heazel, C. OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard. Reference Number: 20-010. 2021. Available online: http://www.opengis.net/doc/IS/CityGML-1/3.0 (accessed on 21 February 2024).
8. Biljecki, F.; Kumar, K.; Nagel, C. CityGML Application Domain Extension (ADE): Overview of developments. *Open Geospat. Data Softw. Stand.* **2018**, *3*, 13. [CrossRef]
9. Van den Brink, L.; Stoter, J.; Zlatanova, S. UML-Based Approach to Developing a CityGML Application Domain Extension: UML-Based Approach to Developing a CityGML Application Domain Extension. *Trans. GIS* **2013**, *17*, 920–942. [CrossRef]
10. Kutzner, T.; Chaturvedi, K.; Kolbe, T.H. CityGML 3.0: New Functions Open Up New Applications. *PFG—J. Photogramm. Remote Sens. Geoinf. Sci.* **2020**, *88*, 43–61. [CrossRef]
11. Biljecki, F.; Lim, J.; Crawford, J.; Moraru, D.; Tauscher, H.; Konde, A.; Adouane, K.; Lawrence, S.; Janssen, P.; Stouffs, R. Extending CityGML for IFC-sourced 3D city models. *Autom. Constr.* **2021**, *121*, 103440. [CrossRef]
12. Saeidian, B.; Rajabifard, A.; Atazadeh, B.; Kalantari, M. A semantic 3D city model for underground land administration: Development and implementation of an ADE for CityGML 3.0. *Tunn. Undergr. Space Technol.* **2023**, *140*, 105267. [CrossRef]
13. Saeidian, B.; Rajabifard, A.; Atazadeh, B.; Kalantari, M. Managing underground legal boundaries in 3D—Extending the CityGML standard. *Undergr. Space* **2024**, *14*, 239–262. [CrossRef]
14. Utility Network ADE for CityGML 3.0 [Git Repository]. Available online: https://github.com/tum-gis/citygml3-utility-network-ade (accessed on 21 February 2024).
15. Bachert, C. Mapping the Energy ADE to CityGML 3.0. MSc. Thesis, Delft University of Technology, Delft, The Netherlands, 20 January 2023. Available online: http://resolver.tudelft.nl/uuid:d253b343-7c96-45ee-9239-5c85594ad4fa (accessed on 21 February 2024).
16. FME Conversion [Website]. Available online: https://hub.safe.com/publishers/con-terra-lab/templates/convert-citygml-2-0-to-3-0 (accessed on 21 February 2024).
17. Chaturvedi, K.; Kolbe, T. Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2016**, *IV-2/W1*, 31–38. [CrossRef]
18. Energy ADE for CityGML 3.0 [Git Repository]. Available online: https://github.com/tudelft3d/EnergyADEv1_toCityGMLv3 (accessed on 21 February 2024).