CityGML

energyADE 3.0

Beta 7

Specifications

Last updated on 26 October 2025

DUT Driving Urban Transitions

DigiTwins4PEDs

The latest version of this document can be downloaded from:
https://github.com/tudelft3d/Energy_ADE

Author

Giorgio Agugiaro
3D Geoinformation Group
Department of Urbanism
Faculty of Architecture and Built Environment
Delft University of Technology
The Netherlands
g.agugiaro@tudelft.nl
https://3d.bk.tudelft.nl/gagugiaro/

Funding

Development of the new Energy ADE has started in spring 2024 within the European DigiTwins4PEDs project, as part of the Work Package 3. The project is funded by the European Commission under the Horizon Europe Partnership scheme. The DUT Call 2022 also contributes to the Urban Transition Mission of Mission Innovation as part of the MICall 2022 initiative.

Acknowledgements

The author would like to thank the following people for their suggestions and fruitful discussions that have helped design the current version of the Energy ADE:

- Rushikesh Padsala and Zhihang Yao (HFT Stuttgart, Germany)
- Thomas Kolbe (TU Munich, Germany)
- Jérôme Kämpf (Idiap, Switzerland)
- Camilo León-Sánchez (TU Delft, The Netherlands)

Particular gratitude is due to Daniele Gasparini for designing the Energy ADE 3.0 logo.

Index

Foreword

This document contains the results of the work started within the European project DigiTwins4PEDs[1] ("Utilisation of Urban Digital Twins to co-create flexible positive energy systems for districts"), in the context of Work Package 3 ("Extension of the Urban Digital Twin Concept with flexibilization strategies"), in particular Task 3.2 on "Extension and update of the CityGML Energy ADE".

To support the project's case studies, the project DigiTwins4PEDs plans to use Urban Digital Twins (UDTs), which is a unified visual interface of integrated information (covering both spatial and non-spatial data, as well as both static physical urban objects and dynamic urban processes), to develop advanced geospatial applications and tackle technical, spatial, regulatory, legal, financial, environmental, social and economic aspects needed to establish PEDs.

UDTs are digital representations of the real world/built environment at the urban scale, but with the granularity that reaches the single building level. For semantic 3D city models, the most relevant, international *open* standard is CityGML[2] by the Open Geospatial Consortium[3] (OGC). CityGML datasets store geometry and semantical information of common city objects such as buildings, trees, roads, land use, etc., found in real-world built environment.

To enhance the capabilities of a semantic 3D city model to cope with energy-related data, and to be better coupled with energy-simulation tools, in the past years an extension to the CityGML standard, called Energy Application Domain Extension (Energy ADE), was developed. The Energy ADE for CityGML is the result of more than 5 years of joint development among different experts from different disciplines (IT, energy, simulation, urban energy planning, municipalities, etc.). The purpose was to define a standardised data model to allow for data interoperability and seamless data exchange between 3D city models (based on the CityGML 2.0 standard) and heterogeneous energy simulation tools and applications. Version 1.0 was released in 2018 as an open data model and has been since used in several projects and applications[4].

Still, its integration in the context of participatory energy planning and PEDs is new and needs to be further explored and improved. However, so far it represents the best candidate and most obvious option to enrich the UDT capabilities in the context of DigiTwins4PEDs.

One of the goals of the DigiTwins4PEDs project is to update and extend the Energy ADE to a new version, which we will refer to as the Energy ADE 2.0, in order to cope with newer challenges and needs related, for example, to PEDs, integration of renewables, and electric mobility.

[1] https://digitwins4peds.eu
[2] https://www.ogc.org/publications/standard/citygml
[3] https://www.ogc.org
[4] https://3d.bk.tudelft.nl/gagugiaro/pdf/2020_06_Energy_ADE_OGC.pdf

1 Introduction

When it comes to the modelling of the built environment, only a relatively small number of *open* standards exist, the common goal being to ease interoperability between heterogeneous software platforms. At urban scale, *semantic 3D city models* (Gröger and Plumer, 2012) have steadily gained momentum in the last 15 years as they offer an integrated and harmonised information hub for a theoretically unlimited number of geospatial application. An integrated virtual city model can represent coherently all city entities, spatially and (possibly) in 3D, together with all relevant characteristics, relations and dependencies. What is more, it can reduce the effort in terms of data preparation and provision, and, secondly, offer clear data structures, ontologies and semantics to facilitate data exchange between different domains and applications.

In the domain of the international data models at urban scale, CityGML is an open data model and information exchange standard by the Open Geospatial Consortium (OGC), defining classes and relations for the most relevant 3D topographic objects in cities (e.g. buildings, transportation infrastructures, city furniture, water bodies) regarding their geometry, topology, semantics, and appearance. The representation of buildings includes, for example, semantically distinct geometries for basement slabs, wall and roof surfaces, as well as specific attributes for the description of the roof shape, usage type, number of storeys, construction year, and building height.

When it comes to modelling energy-related objects and properties, however, even CityGML, which has been intentionally conceived as an application-independent information model, partially falls short: some attributes like the year of construction, the building class and usage are provided, but, overall, they are too few. Additional attributes can indeed be defined as *generic attributes* by means of the so-called Generics module, but they cannot be stored natively in a systematic and semantically standardized way. Nevertheless, CityGML can be extended by means of so-called Application Domain Extensions (ADE): depending on the specific needs, new features or properties can be added, hence greatly augmenting modelling capabilities offered by CityGML.

Work to extend CityGML in order to tackle the aforementioned shortcomings and provide a standardised data model to deal with energy-related data required for *Urban Energy Modelling* (UEM) started in 2013 at TU Berlin and HFT Stuttgart first. From 2014, an international, consensus-based consortium was set up that took care of developing what became known as the CityGML Energy Application Doman Extension, or, in short, Energy ADE. From now on, it will be called Energy ADE *1.0*, in order to distinguish it from the Energy ADE 2.0, which is the main subject of this document.

1.1 The Energy ADE 1.0

The Energy ADE 1.0 (Agugiaro et al., 2018) was created as an extension to the international open standard CityGML for semantic 3D city models. The Energy ADE 1.0 is set in the larger picture of Urban Energy Modelling (UEM), which has been experiencing a steady increase in terms of popularity, development efforts and adoption in the last decade, in order to support the energy transition process at city scale.

The Energy ADE 1.0 was developed to offer a unique and standard-based data model to allow for both detailed single-building energy simulation (based on sophisticated models for building physics and occupants' behaviour) and city-wide, bottom-up energy assessments, with particular focus on the buildings sector (see Figure 1). The Energy ADE has been conceived to offer a standardised and omni-comprehensive urban data model covering also the energy domain. The Energy ADE 1.0 was released in 2018 as the results of an open, consensus-based development process that started in 2013 and that comprised several international parties and disciplinary fields (from computer sciences to energy specialists).

Figure 1. Example of energy modelling at building scale [left] und district scale [right].

The Energy ADE 1.0 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 here:

- The Energy ADE **Core module** (in light pink) defines additional attributes for the CityGML core::*_CityObject* and bldg::*_AbstractBuilding* classes. It also provides new abstract base classes for the other modules and establishes additional data types and enumerations
- 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 **Occupants' 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 **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 **Materials and construction module** (in blue) enables the modelling of the composition of construction surfaces through different layers and their physical properties
- 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.

A complete description of the Energy ADE, its overall structure, as well as its classes is provided by Agugiaro et al. (2018). Further resources, such as the UML diagrams, the XSD file and test datasets can be found online at the corresponding CityGML Wiki webpage[5]. It is worth mentioning that from the full Energy ADE 1.0 a reduced, simplified profile was extracted. It is called "KIT Energy ADE" profile, and it contains a *subset* of the Energy ADE 1.0.

[5] https://www.citygmlwiki.org/index.php/CityGML_Energy_ADE

Finally, the Energy ADE 1.0 data model, although originally developed for CityGML 2.0, has been recently mapped to CityGML 3.0, the latest, currently available version of the CityGML standard released in 2021. Further details can be found in Bachert et al. (2024). The mapping to CityGML 3.0 has allowed, among the rest, to collect experience on the modelling approach for ADEs in the case of CityGML 3.0, but also to evaluate how the new CityGML 3.0 standard offers new classes and functionalities that make (part of) the Energy ADE 1.0 redundant.

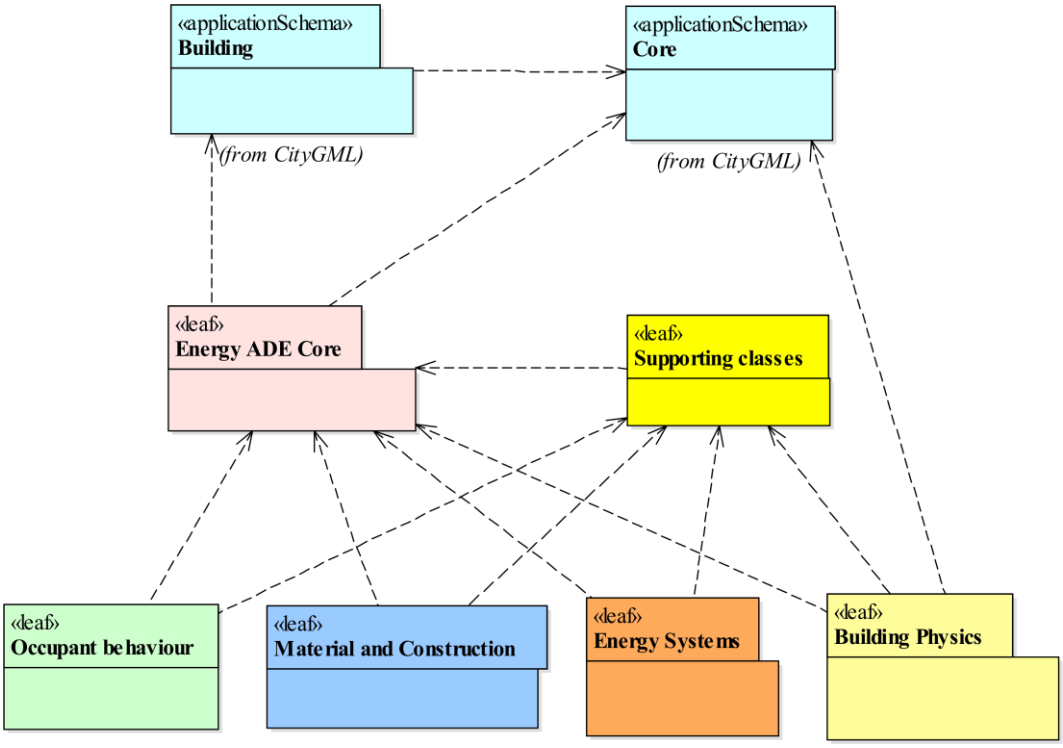


Figure 2. Package overview of the Energy ADE 1.0 for CityGML 2.0. Image taken from Bachert et al. (2024).

1.2 The Energy ADE 2.0

After the release of of the Energy ADE 1.0, preliminary work was carried out to start the development of the Energy ADE 2.0, which was eventually released in 2019.
As a matter of fact, the Energy ADE 2.0 has very minor differences from the previous version, and they consist mainly in the correction of minor modelling errors (e.g. in the WeatherStation and WeatherData classes) and minor changes in the Contruction, Occupancy, and TimeSeries modules. The main difference is that the whole Device system has been dropped, the intention being to design a new, more simplified one. However, after the its release in 2019, the Energy ADE has essentially stopped. Except for some internal development carried out at the Karlsruhe Institute of Technology, this version has never been used elsewhere. Further information can be found on the respective CityGML Wikipage[6].
Due to its limited differences from the Energy ADE 1.0, and its lack of further development and publicly available implementation, this version will not be further mentioned in this document.

[6] https://www.citygmlwiki.org/index.php?title=CityGML_Energy_ADE_V._2.0

2 The Energy ADE 3.0

2.1 Motivation and goals

The development of the Energy ADE 3.0, like the previous versions, has been driven by the following two main objectives:

- To store and manage energy-relevant data collected at urban scale, such as building usage, year of construction, number of dwellings and residents, etc.
- To provide data to assess the energy performance of buildings using a variety of different approaches and software tools, e.g. ranging from standard-based energy balance methods as of ISO 13790, to sub-hourly dynamic simulations by means of specific simulation software programs.

Additionally, these goals have also been considered:

- To improve the data model based on the experiences collected since 2018 thanks to the use and implementation of the Energy ADE 1.0 in several national and international projects. This means to partially simplify, and to partially extend the Energy ADE 1.0 when it comes to classes, properties and associations between classes
- To update and enhance the data model in such a way that it will ease the future conversion/mapping of the Energy ADE 3.0 to CityGML 3.0, also drawing from the experiences and the results of the mapping carried out on the Energy ADE 1.0 towards CityGML 3.0. Given the expected slow but steady transition of data and tools from CityGML 2.0 to CityGML 3.0, it is reasonable to conceive a new data model that can be ported to CityGML 3.0 without major issues
- Finally, to keep compatibility with the KIT profile of the Energy ADE 1.0 data, although some data conversion may be required. Please note that keeping the compatibility with the full Energy ADE 1.0 is *not* a goal. Please refer to section 11 for more details.

2.2 Overview of the development process

The methodology to develop the Energy ADE 3.0 follows a rather well-known and established approach, which has been already described and published by van den Brink et al (2013). The definition of the conceptual model of the Energy ADE 3.0 is complemented with the generation of the XSD (XML Schema Definition) file[7], the creation of test data, and the eventual implementation of conversion tools, as well as a database implementation.
Overall, the developed methodology consists of three steps which are summarised in Figure 3. Following a model-driven approach, first the 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 ADE. The UML diagrams are created using the

[7] https://en.wikipedia.org/wiki/XML_Schema_(W3C)

modelling software Enterprise Architect[8] by Sparx Systems. This first step is shown in Figure 3 in green.
In the second step, depicted in yellow, the transfer format is derived from the UML data model. Here, based on the GML target encoding, an XSD file is derived. It specifies how to correctly read, write and validate Energy ADE 3.0 GML files for CityGML 2.0. The XSD schema is created using the Java tool ShapeChange[9], which requires a specific configuration file and then automatically applies the encoding rules to the UML class diagrams.
The last, third step, depicted in light azure, consists in creating CityGML 2.0 + Energy ADE 3.0 test datasets which will be used to test the conversion tools developed in the following step. The test datasets are created, for example, using Safe Software's FME Form[10] and the XSD file previously obtained. The creation of the datasets has the additional advantage to allow for immediate testing of the developed data model, and to adjust it, if necessary, resulting actually in an iterative development process.
Finally, the development of the conversion tools encompasses the development of Java-based libraries that are needed, on the one hand, to load and save XML-based CityGML files with Energy ADE 3.0 content, and, on the other hand, to convert such XML-based files to a SQL-based encoding, therefore allowing to import and export data from/to the CityGML 3D City Database[11].

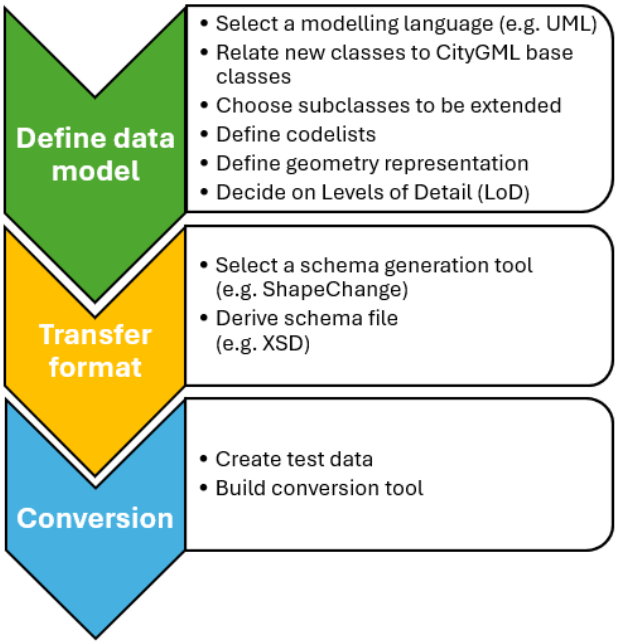


Figure 3. Schematic workflow to develop the Energy ADE 3.0. Image adapted from Bachert et al. (2024).

The workflow presented in Figure 3 has been further refined for the specific needs and tasks of the DigiTwins4PEDs project. The detailed workflow is shown in Figure 4.

The following sections of this document will refer to the latter workflow and will be explained in the respective sections and subsections. Section 3 describes the Energy ADE 3.0 conceptual

[8] https://sparxsystems.com
[9] https://shapechange.net
[10] https://fme.safe.com
[11] https://www.3dcitydb.org

data model. It also mentions the major changes with regard to the Energy ADE 1.0 and the underlying motivations. Similarly to the Energy ADE 1.0, it will follow a module-based approach for the documentation. Section 4 provides some modelling rules and guidelines to help when creating and writing CityGML datasets with Energy ADE 3.0 contents. Section 5 briefly describes the process of deriving the XSD file from the UML model. Section 6 provides an overview of the test data that can be used for testing and learning purposes, while section 7 describes how the generation of the DDL scripts was carried out to extend the CityGML 3D City Database. Finally, section 8 briefly describes a workbench for FME Form that allows to import Energy ADE 3.0 data into the 3D City Database. The workbench has been developed for testing purposes: it allows to test the current implementation of the database schema for the Energy ADE 3.0 and, at the same time, it permits to interact with the database while the Java-based libraries – which are needed to add Energy ADE 3.0 support to the 3D City Database – are still in development.

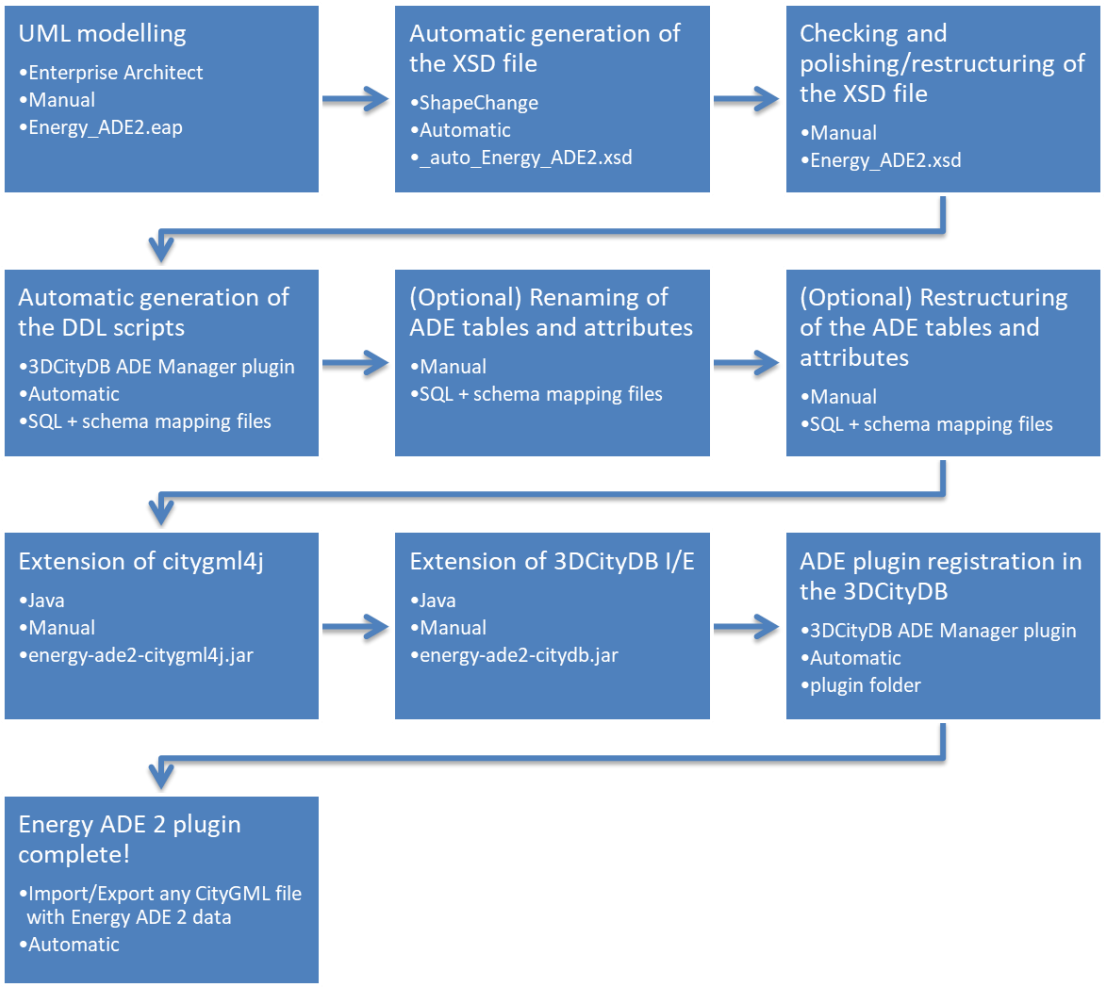


Figure 4. Detailed workflow of the development of the Energy ADE 3.0 within the DigiTwins4PEDs project.

3 Conceptual data model

3.1 Introduction

The Energy ADE 3.0 builds upon and expands the Energy ADE 1.0. It is meant to improve the usability and the modelling capability of the previous version, also considering the experiences from real-world implementations accumulated in the years since it release.
Just like its predecessor, the Energy ADE 3.0 builds upon CityGML modules (namely, the Core, the Buildings, and the CityObjectGroup ones) by extending some of their classes, while it adds its own classes, enumerations and codelists. Again, it is structured in a modular way and Figure 5 shows the overview of the Energy ADE 3.0 packages while depicting the dependencies between the different modules. The thematic modules are briefly described here:

- The **Core module** (in light pink) defines additional attributes for the CityGML core::*_CityObject* and bldg::*_AbstractBuilding* classes. It also provides new abstract base classes for the other modules
- The **Building physics module** (in yellow) defines additional attributes for the CityGML bldg::*_BoundarySurface* and bldg::*_Opening* classes. Additionally, it defines new classes to model the thermal hull of a building
- The **Occupancy module** (in green) defines classes to model different usage zones and how they are utilised by occupants. By including schedules, it is possible to represent their behaviours
- The **Devices module** (in orange) provides classes to model different types of energy-related devices that are responsible for transforming or storing energy (e.g. solar collectors, storage devices, etc.);
- The **Urban function areas module** (in dark green) allows to define spatial entities that allow to aggregate values
- The **Weather station module** (in dark blue) contains a class to model weather stations
- The **Layered construction module** (in blue) enables the modelling of the composition of construction surfaces through different layers and their physical properties
- The **Resources module** (in light violet) allows to model resources (energy, water, food, etc.) that any urban object may demand, produce or store
- The **Schedules module** (in dark pink) comprises classes for different types of schedules
- The **Time series module** (in light yellow) comprises specific classes for different types of time series. They are used to add time-dependent values to the other module parameters
- The **CodeLists** (in white), **Enumerations** (in dark grey) and **DataTypes** (in light orange) **modules** contain support classes that are needed by the other modules.

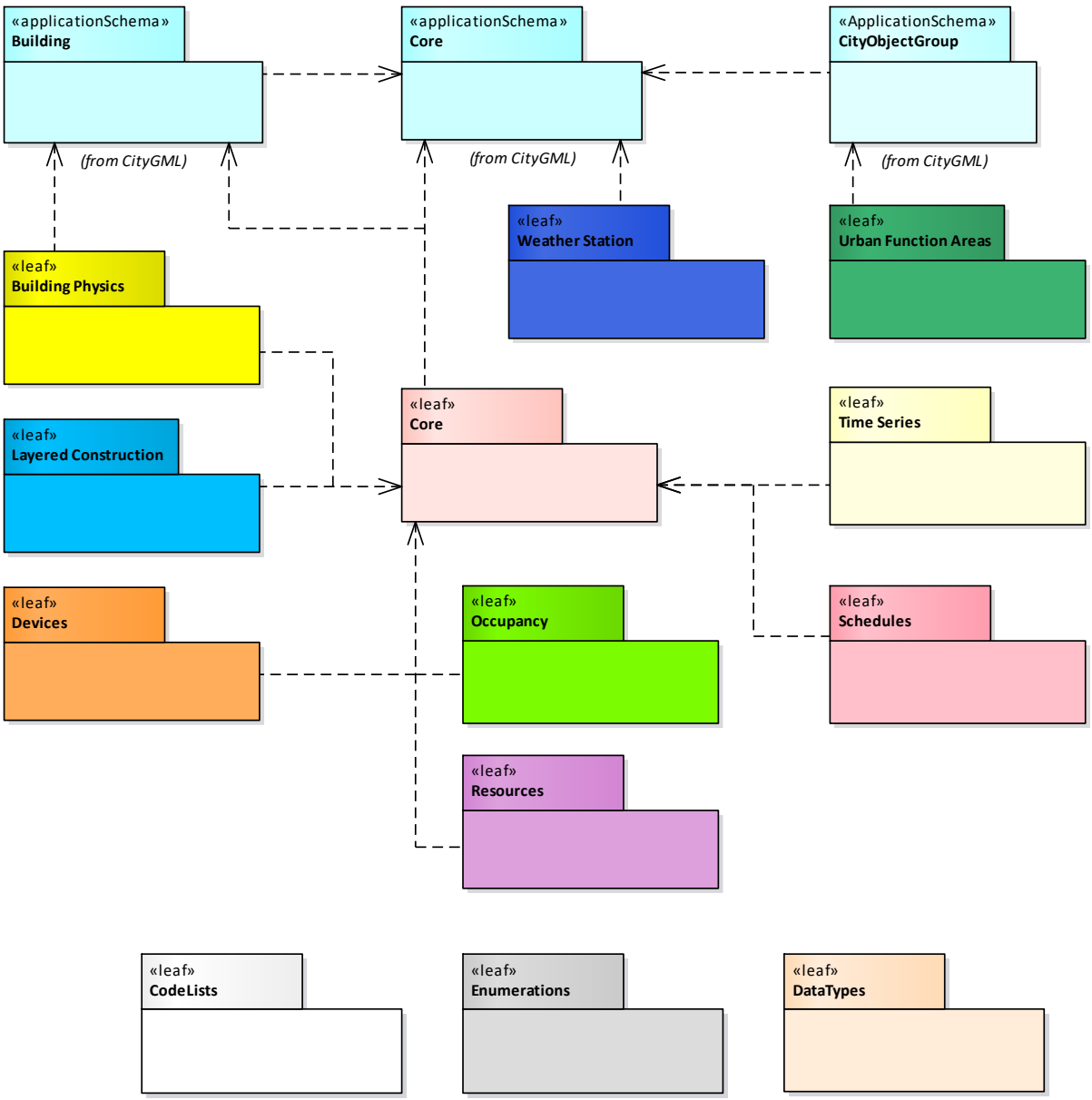


Figure 5. Package overview of the Energy ADE 3.0 for CityGML 2.0. The colours representing the different packages will be used throughout this document. Dependencies between the different packages are represented by means of dashed arrows, except for the packages CodeLists, Enumerations, and DataTypes, for which the dependencies are omitted for better readability.

The Energy ADE 3.0 takes also inspiration from new ADE developments in the CityGML "world" since 2018, such as:

- the **i-Urban Revitalization ADE** for CityGML 2.0, or **i-UR ADE** in short (Akatoshi et al., 2020)
- the **Food-Water-Energy ADE** for CityGML 2.0, or **FWE ADE** in short (Padsala et al., 2021)
- the **Utility Network ADE** for CityGML 2.0, or **UN ADE** in short (Kutzner et al., 2018)
- **CityGML 3.0**, in particular the Core and the Dynamizer modules (Chaturvedi and Kolbe, 2016; Kutzner et al., 2020; OGC, 2021).

Some examples of the influence from the above mentioned ADEs are the new class *UrbanFunctionArea*, that fundamentally coincides with the homonymous class in the i-UR ADE

(see section 3.6). Another example is the major overhaul of class *EnergyDemand* by means of the classes belonging to the Resources module (see section 3.9), which takes inspiration from and further develops some concepts of the Food-Energy-Water ADE. The Utility Network ADE has inspired the new class *UtilityNetworkConnection*, which is meant to describe the connectivity of any city object when it comes to the different utility networks, even if the network layout itself is not available, e.g. in terms of geometrical and/or topological information. From CityGML 3.0, the *CityObjectRelation* association class has been "backported" to the Energy ADE 3.0, as, for example, it allows to model in an easier way the adjacency relation between two buildings, or the relation between the shared walls (party walls) between two buildings, or between two thermal zones. Additionally, the class stereotype «type», used several times in the Energy ADE 1.0, has been replaced because it is not used anymore in CityGML 3.0. This will ease, in future, the porting of the Energy ADE 3.0 from CityGML 2.0 to CityGML 3.0.
The coming sections provide a description of the main characteristics of each module, as well as an overview of the main changes between the Energy ADE 1.0 and the Energy ADE 3.0 **beta7**, the latest version at the moment of writing. Whenever necessary, the reader is invited to have a UML diagram of the Energy ADE 1.0 at hand to facilitate the comparison between the two Energy ADE versions.
Please note that the colours presented in the package diagram depicted in Figure 5 are also adopted in the remainder of this document for better readability. Classes belonging to the ADE modules are represented using the same colours, while all classes used for CityGML are always depicted in light cyan.
The PDF file containing the complete UML diagrams of the Energy ADE 1.0 is available on the CityGML Wiki Energy ADE Webpage[12], while the complete UML diagram of the Energy ADE 3.0 is available for download from the GitHub repository[13] of the 3DGeoinformation group of TU Delft.

3.2 The Core module

3.2.1 Characteristics

As the name says, the Core module represents the central part of the Energy ADE 3.0. Its overview in terms of UML diagrams is presented in Figure 6 and Figure 7. The Core module has two main functions:

- It extends two classes of the CityGML standard with energy relevant properties (namely the *_CityObject* and the *_AbstractBuilding* classes)
- It provides abstract base classes for the other modules of the Energy ADE 3.0 (*AbstractLayeredConstruction*, *AbstractLibrary, AbstractDevice, AbstractResource*, *AbstractSchedule, AbstractTimeSeries*). These abstract base classes will be described in the sections covering the respective modules, although they are contained in the Core module. In order to additionally hint at the module they are connected to, they are coloured differently in Figure 6 and Figure 7 instead of the "default" light pink.

[12] https://www.citygmlwiki.org/images/f/fb/UML-Diagrams_EnergyADE.pdf
[13] https://github.com/tudelft3d/DigiTwins4PEDs/tree/main/energy_ade_2

Thanks to this structure, it is possible to avoid too many mutual dependencies between the modules.

The CityGML abstract base class *_CityObject* is extended by means of the ADE hook mechanism. In terms of geometry, the ADE *_CityObject* class adds the possibility to optionally represent any city object by means of a point geometry via the *referencePoint* property. This change reflects and "backports" what introduced in CityGML 3.0. Additionally, a generic concept of suitability is added to any city object by means of the property *suitability*. The corresponding datatype *Suitability* (in package DataTypes) allows to define and store any suitability reason, a suitability value expressed as a decimal number in [0..1], and to optionally define temporal properties for such suitability through a link to a schedule. The idea is to be able to store and retrieve information about a certain suitability that may be the result of a simple (or complex) analysis and selection process (e.g. suitability for installation of photovoltaic panels, or for green roofs, etc.).

The *CityObjectRelation* association class has been backported from CityGML 3.0 to facilitate the modelling of additional relations between city objects. For example, class *CityObjectRelation* can be used to store adjacency relations between buildings even in the (rather common case) case that, due to geometrical errors or inaccuracies, such spatial relationships cannot be computed directly from the LoD0 footprint geometries. Reason for it could be for example gaps or overlaps between two footprint polygons. Another concrete application example is the computation of party walls between adjacent buildings which sometimes fails due to the afore mentioned problems. Similarly, adjacency relations can be now modelled also between thermal zones, or, in a similar way, correspondences between thermal zones and other CityGML classes (e.g. *Room*).

Besides the relations to *AbstractResource*, *AbstractLayeredConstruction*, *AbstractDevice*, the ADE *_CityObject* class contains also relations to other classes, such as:

- *WeatherData*. Every city object can have an arbitrary number of *WeatherData* objects, each of them characterised by the attribute *type* (e.g. temperature, precipitation, wind, etc.). Actual values can be stored either as yearly values (attribute *yearlyValue*) or be associated with a time-dependent object derived from class *AbstractTimeSeries* (through relation *timeDependentValues*). The *valueType* attribute can be used to further specify whether the weather parameter (e.g. temperature) is for example the maximum, minimum, average, etc. Additionally, through the *position* attribute, a point geometry can be defined to specify the position the *WeatherData* object refers to. Finally, the *libraryCode* attribute can be used to store a specific code (with the associated code space) for a weather parameters to be looked up in an external database.
- Class *UtilityNetworkConnection* is used to model the actual or potential connectivity status of a city object with regard to any utility network, be it gas, electricity, drink water, sewage, etc. (attribute *networkType*). By means of the attribute *connectionStatus*, not only the current connectivity status can be modelled, but also the future one in case, for example, the building is disconnected from the gas network and opts for a complete electrification. If specific information about the network is available, this can be stored using the *networkID* and the *networkNodeID* attributes (e.g. linking to network-related objects of the Utility Network ADE, or any

other external resources). Finally, attributes *functionInNetwork* and *usageInNetwork* can be used to further specify the connection to the network. Many of these attributes are taken over from a conceptually similar class in the Utility Network ADE.

An advantage of using class *UtilityNetworkConnection* is that the connectivity status can be modelled regardless of the availability of actual network data (be it topological or geographical) – which is a very common problem with "real-world" datasets due to the complexity of collecting such data. At the same time, as class *UtilityNetworkConnection* takes inspiration from the Utility Network ADE, it can be seen as a connecting point between the two ADEs.

Three more abstract base classes are visible in Figure 6: *AbstractSchedule* (which will be described in section 3.10), *AbstractTimeSeries* (see section 3.11) and *AbstractLibrary*. The latter represents an abstract base class that will be used to derive specific library classes used as containers, e.g. for *LayeredConstruction* or *Material* objects. The main idea of a Library object is to act as a container for those objects that are generally coming from libraries and not measured directly from the real world. In the case of UBEM, this happens frequently, for example, when characterising the physical properties of the (thermal) envelope elements of a buildings, as such surveyed data are generally not available at building level. In this sense, the possibility to model and store library data within a unique container, and having such library data referenced by those elements requiring it, is a convenient solution, which is also used in common energy simulation software packages.
In the specific case of CityGML, class *AbstractLibrary* is meant to be a top-level element, derived from base class *_CityObject*, although it would make more sense to derive it from *_Feature* – as it is the case of CityGML class *Appearance*. This is, at the moment, a decision taken for pragmatic reasons, as this simplifies the management of such library objects, e.g. by means of the 3DCityDB Importer/Exporter that allows to choose which top-level city objects to import or export from/to the 3D City Database.

When it comes to the CityGML class *_AbstractBuilding*, this is also extended by means of the ADE hook mechanism. The ADE *_AbstractBuilding* class contains several energy-relevant attributes at building level. Please note that in the UML diagrams the prefix "*bdg*" is used with class property names to avoid naming conflicts in the resulting XSD file, but – for the sake of readability – in this document they will be referred to without the prefix.
In class ADE *_AbstractBuilding*, attribute *type* is intended to store information about the typology of the building (e.g. single-family house, terraced house, etc.). The thermal status of the attic and the cellar can be specified by means of the attributes *basementThermalStatus* and *atticThermalStatus*, as well as information about the rough classification of the building construction structure (*constructionWeight*), its protection status (e.g. in case of historical buildings), as well as relevant dimensional characteristics (by means of qualified attributes *height*, *area*, and *volume*). If a building possesses Energy Performance Certificates, or information is known about its refurbishment history, such information can be stored using the *EnergyPerformanceCertificate* and the *RefurbishmentMeasure* classes, respectively.
Finally, for more advanced energy simulations, concepts such as thermal zones, and usage zones must be introduced. For this reason, the ADE *_AbstractBuilding* class is related to the abstract base classes *AbstractThermalZone*, *AbstractUsageZone*, and *AbstractBuildingUnit*. The former will be described in the Building physics module, while the latter two will be

described in the Occupancy module. It is however relevant to mention here that all these three abstract classes are derived from the *AbstractBuildingPartition* class, which allows to store information about area and volume of each partition (attributes *area* and *volume*) and up to three geometrical representations by means of solid geometries (in LoD1, LoD2, and LoD3, respectively). The *AbstractBuildingPartition* takes inspiration from CityGML 3.0 *AbstractBuildingSubdivision*.

3.2.2 Main changes from the Energy ADE 1.0

When it comes to the changes and additions compared to the Energy ADE 1.0, they can be listed as follows:

- Every city object can now be represented also by a point geometry. In the Energy ADE 1.0 this was possible only for the classes *Building* and *BuildingPart* (i.e. derived by ADE *_AbstractBuilding*)
- The association between *AbstractDevice* (previously named *AbstractEnergySystem*) and *_CityObject* has been inverted in terms of navigability. Now it goes from *_CityObject* to *AbstractDevice*, while in the Energy ADE 1.0 it went from *AbstractEnergySystem* to *_CityObject* (and was named *installedOn*). Now, every *_CityObject* can have 0..* *AbstractDevice* objects
- Class *AbstractDevice* has been simplified when it comes to property *heatDissipation*. Instead of referring to class *HeatExchangeType* (which has been dropped), all properties are now within the class (*heatDissipationConvectiveFraction*, *heatDissipationLatentFraction* and *heatDissipationRadiantFraction*). The original property *heatDissipation* is now of type Measure and is meant to contain the average value of heat dissipation
- The association between *AbstractDevice* and *DeviceOperation* (previously named *OperationSchedule*) has been renamed from *operationSchedule* to *deviceOperation*
- Every city object can have 0..1 *AbstractLayeredConstruction* objects. In the Energy ADE 1.0, only the ADE *_AbstractBuilding* class, as well as *ThermalBoundary* and *ThermalOpening* classes were associated to an *AbstractConstruction*. Please note: All *Construction* classes have now been renamed to *LayeredConstruction*
- The *EnergyDemand* class has been dropped and substituted by the *AbstractResource*. More details are provided in the section describing the Resource module
- When it comes to class ADE *_AbstractBuilding*, there have been only minor changes. For example, the types associated to the reference height, the floor area, and the volume are now of type *QualifiedHeight*, *QualifiedArea*, and *QualifiedVolume*, respectively. This renaming follows, among the rest, the same logic as in CityGML 3.0
- Class *AbstractBuildingPartition* is new and it allows for the usage of solid geometry (from LoD1 to LoD3). The addition of the LoD concept is new to the Energy ADE 3.0 and will be further explained in the Building physics and Occupancy modules
- The classes for *EnergyPerformanceCertificate* and *RefurbishmentMeasure* have been slightly improved. In particular, the class *RefurbishmentMeasure* has a new attribute *libraryCode* which allows to store a code and, possibly, the codespace associated to it, in case there is a library or an external database containing a set of standardised/typical refurbishment measures

- The *AbstractLibrary* class is new, as well as the whole concept of library top-level containers.

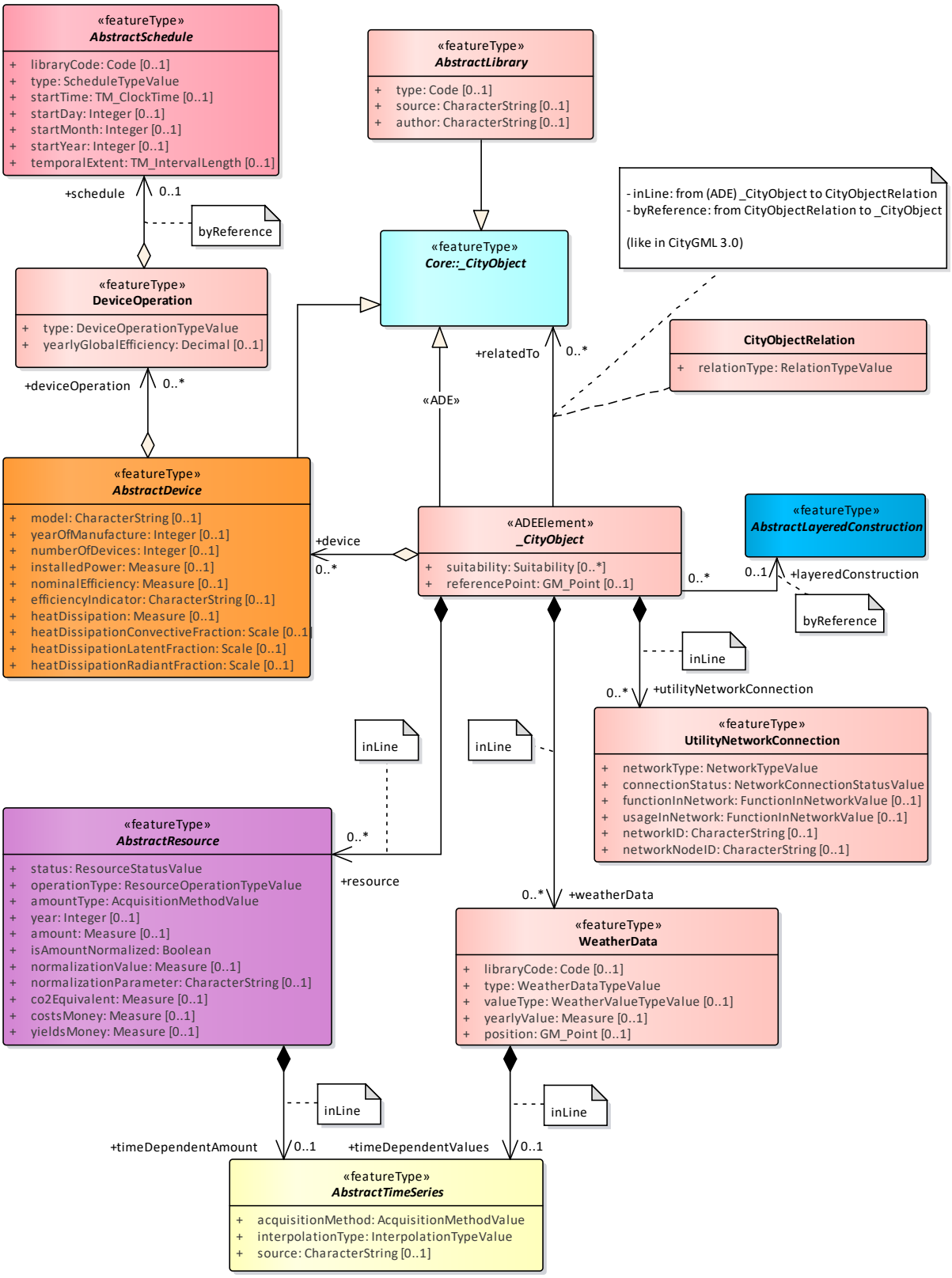


Figure 6. Detail view of the Energy ADE 3.0 Core module, focusing on the *_CityObject* class and its dependent classes.

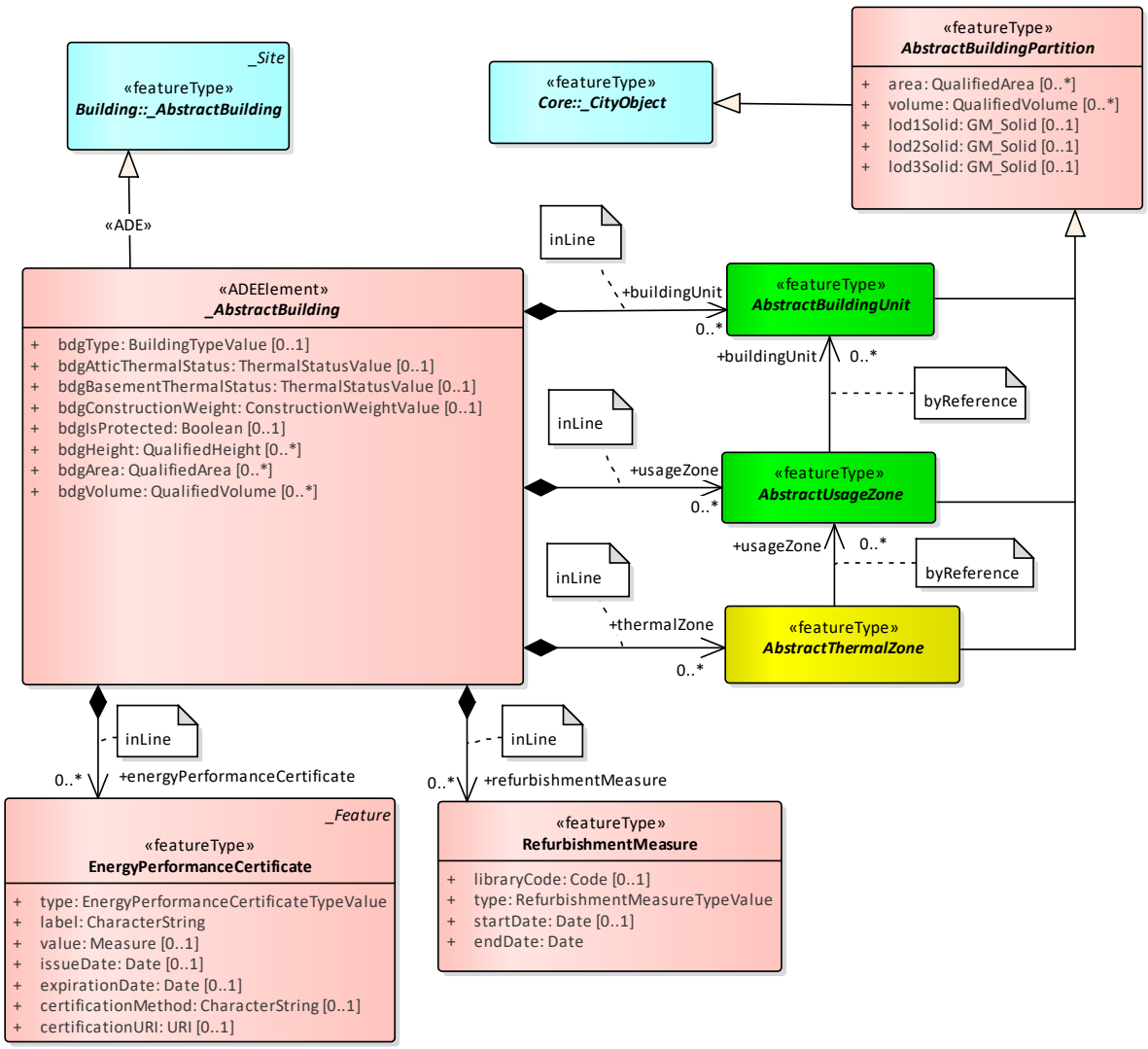


Figure 7. Detail view of the Energy ADE 3.0 Core module, focusing on the ADE *_AbstractBuilding* class and its related classes.

3.3 The Building physics module

3.3.1 Characteristics

The Building physics module provides classes for detailed simulations of a building thermal behaviour, e.g. for computation of space heating and space cooling demands. The Building physics module has the *ThermalZone* as its principal class, which represents the reference volume for heating and cooling demand calculation. It is generally a “thermally homogeneous” space considered as isothermal, and, for simplified building energy modelling, it may also refer to several building rooms. A building may be separated into several *ThermalZone* objects.

Class *ThermalZone* possesses energy-related attributes characterising its conditioning status (*isCooled*, *isHeated*) and overall building physics properties (*heatCapacity*, *infiltrationRate*). Finally, two attributes (*coincidesWithLoD2Hull* and *coincidesWithLoD3Hull*) are used to specify when the thermal zone is exactly bounded by the geometries (via CityGML thematic surfaces) that define the building hull. More details on the usage of these two attributes are provided in Section 4 about modelling rules and guidelines.

If occupied, a *ThermalZone* object contains one or more *UsageZone* objects, each one providing information about the usage for the heating and cooling demand calculation (see Section 3.4 for the Occupancy module). Inherited via its parent class *AbstractBuildingPartition*, a *ThermalZone* has also attributes regarding its dimensions (*area* and *volume*) and its geometrical representation via solid geometries in different LoDs, from LoD1 to LoD3 (but not in LoD4!).
For LoD2 and LoD3, *ThermalZone* objects can be additionally represented geometrically by means of thematic surfaces, in a conceptually similar fashion as in CityGML for buildings. For this reason, the *ThermalZone* class has a property named *thermalBoundary*, which mimics the *boundedBy* property of *_AbstractBuilding* in CityGML when it comes to relation to thematic surfaces.
For LoD2 representations, all existing classes for thematic surfaces from CityGML (*WallSurface*, *RoofSurface*, *GroundSurface*, etc.) can be used to compose the envelope representing the *ThermalZone*. Additionally four more specific classes are derived from *_BoundarySurface*: *PartyWallSurface*, *IntermediateFloorSurface*, *BasementCeilingSurface*, and *AtticFloorSurface*. As in CityGML, no openings are allowed in LoD2. For LoD3, as in CityGML, thermal openings are allowed, therefore CityGML classes *Window* and *Door* can be used too. More information on how to use the existing and new classes to compose the hull of a thermal zone can be found in Section 4.3 where the modelling rules and guidelines, specifically for the Building physics module, are presented.
In order to enrich the description of the thematic surfaces, the CityGML *_BoundarySurface* and *_Opening* classes are both extended by means of the ADE hook mechanism. Similarly to the ADE *_AbstractBuilding* class, both ADE classes contain property names that are prefixed with “bdgBdrySurf” and “bdgOpn”, respectively. For the sake of readability, the attribute names will be referred to in the next paragraphs without using these prefixes. In the case of ADE *_BoundarySurface*, most attributes relate to geometrical properties, such as *thickness*, *totalSurfaceArea*, *opaqueSurfaceArea*, *azimuth*, *inclination*, *groundViewFactor* and *skyViewFactor*.
In particular, the attribute *totalSurfaceArea* is meant to store the total, gross value of a surface area, including the area of openings. The attribute to store the area excluding the openings is the *opaqueSurface* area. If, for example when modelling in LoD2, the actual area of the openings is not known, then it can be derived indirectly using the *openingToSurfaceRatio* (with corresponds to the more commonly known *window-to-wall ratio*). Additionally, some energy-specific attributes are also available, such as *heatCapacity* and *isAdiabatic.* In the case of the ADE *_Opening*, a similar, but reduced set of similar attributes have been added.
Thanks to these new attributes, it is now possible to store information about area, inclination, azimuth, etc. for each thematic surface, also for those being used to model *Building* and *BuildingPart* objects. For example, such attributes can be pre-calculated and stored also for *Roof* objects in order to perform a query on values regarding a certain inclination, area, azimuth, etc. The main idea is that, in this way, queries such as "*Select all RoofSurfaces tilted by x degrees and oriented towards y*" can be sped up significantly without having to (re)compute them every time on the fly from the geometries.

3.3.2 Main changes from the Energy ADE 1.0

The Building physics module is one of the modules that has changed the most compared to the Energy ADE 1.0, where specific classes for *ThermalBoundary* and *ThermalOpening* existed, and were decoupled from the CityGML thematic surfaces. However, in the Energy ADE 1.0 the

geometrical representation of classes *ThermalZone*, *ThermalBoundary* and *ThermalOpening* was decoupled from the "standard" CityGML one. Besides, only one representation was possible.

In the Energy ADE 3.0, both classes *ThermalBoundary* and *ThermalOpening* have been dropped. As a consequence, the relation between *ThermalBoundary* and *_BoundarySurface*, and finally between *ThermalOpening* and CityGML *_Opening* have been dropped too, as they are not needed anymore. As a result, the concept of shading devices has been reworked too. A *MovableShadingDevice* class has been added to the Devices module. More details will be provided in that module.

Thanks to the new class *RelationToCityObject* in the Core module, the previous relation between *ThermalZone* and CityGML *Room* has been dropped as it would be redundant. Relations between *ThermalBoundaries, ThermalOpening* and *AbstractConstruction* have also been dropped, as every city object is now related to *AbstractLayeredConstruction* as this relation is inherited from ADE *_CityObject* class.

Finally, the new *ThermalZone* class receives two Boolean attributes: *coincidesWithLoD2Hull* and *coincidesWithLoD3Hull*. These two attributes are meant to identify two special cases, i.e. when a building is modelled as a single thermal zone *and* the thermal boundaries coincide exactly with the LoD2 (or LoD3) hull of the *Building/BuildingPart*. In this specific case (still rather frequent when working at urban scale with UBEM software tools like SimStadt), now it is not necessary anymore to re-model the thermal boundaries of the thermal zone. In the Energy ADE 1.0, this would lead to creating copies of the same objects (the thematic surfaces) that should exist as components both of a *Building*/*BuildingPart* object and of a *ThermalZone* object and therefore represent a redundant duplication of data.

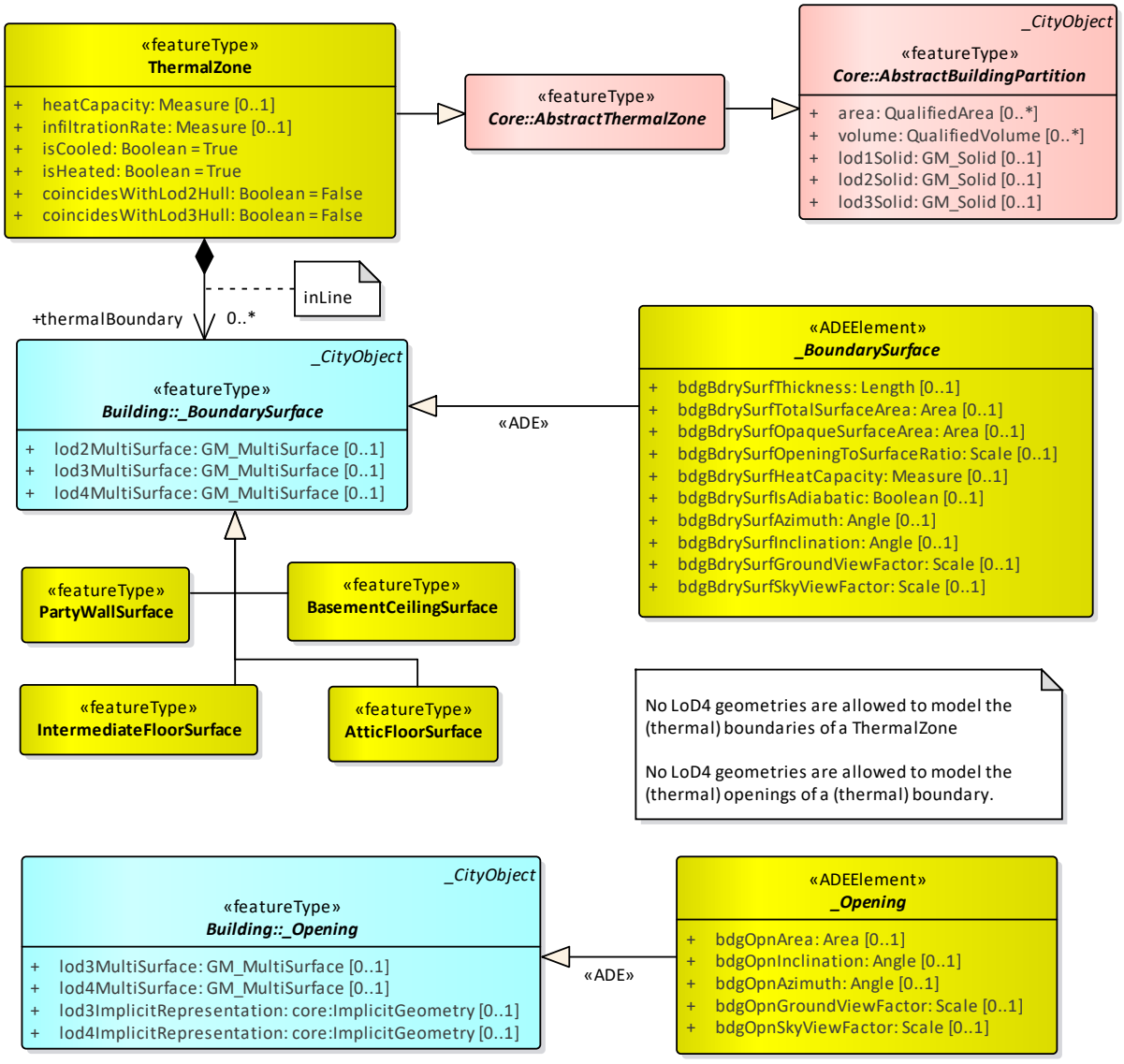


Figure 8. Overview of the Energy ADE 3.0 building physics module.

3.4 The Occupancy module

3.4.1 Characteristics

The Occupancy module contains classes to represent the occupants of a building, as far as it is relevant for energy simulations. The overview of the Occupancy module in terms of UML diagrams is presented in Figure 9.

Class *UsageZone* is used to define a region of homogeneous usage in a building. Such regions can be contained in and referenced by a *ThermalZone*. Additionally, a *UsageZone* can contain one or more *BuildingUnit* objects, and it is used to model the usage for the heating and cooling demand calculation. Besides the properties inherited via *AbstractBuildingPartition* (and already explained in section 3.2 about the Core module), attribute *type* defines the specific usage of that thermal zone (residential, commercial, etc.), while attribute *isPrimary* is used to express whether this is the primary or the most relevant usage of the building. It is possible to define the number of building units that are part of the *UsageZone* object using the attribute

numberOfBuildingUnits, without the need to explicitly model them via the *BuildingUnit* class. Average internal heat gains can be modelled by means of property *internalHeatGains* (and, optionally, further specified in terms of convective, latent and radiant fractions), while the remaining relations (*heatingSchedule*, *coolingSchedule*, *ventilationSchedule*) are used to connect to objects of the Schedule module, through the *AbstractSchedule* abstract base class. Optionally, a *UsageZone* may be subdivided into different *BuildingUnit* objects holding information about the specific use (*type*), the number of rooms (*numberOfRooms*), ownership information (*ownerName*, *ownershipType*). If applicable, a *BuildingUnit* object can be related to its *EnergyPerformanceCertificate* and *RefurbishmentMeasure* object(s). Finally, a relation to an *Address* can be provided too.
Both the *UsageZone* and the *BuildingUnit* classes can be related to class *Occupants*, which can be used for detailed assessment of the internal gains due to the heat produced by people (property *heatDissipation*, and the additional, optional further characterisation in terms of convective, latent and radiant fractions). Class *Occupants* represents a homogeneous group of occupants, categorised in one t*ype* (e.g. residents, workers, visitors etc.), for which the number of occupants can be provided, as well as their occupancy schedule (via attribute *numberOfOccupants* and relation *occupancyRate*, respectively). Additionally, some socio-economic parameters are available (*averageDiet*, *averageIncomeLevel*, *averageInstructionLevel*).

3.4.2 Main changes from the Energy ADE 1.0

When it comes to the differences between Energy ADE 1.0 and 2.0, there have been no significant changes, save for an overall simplification of the module. The main changes are:

- Classes *UsageZone* and *BuildingUnit* have some the attributes reworked, as some of them are inherited by their parent classes (mainly: *AbstractBuildingPartition*). Just like *ThermalZone*, now they can both be modelled via solid geometries from LoD1 to LoD3
- Class *BuildingUnit* is not only associated with *EnergyPerformanceCertificate* (as in the Energy ADE 1.0) but also with a *RefurbishmentMeasure* class
- Class *UsageZone* has been simplified when it comes to property *averageInternalGains*. Instead of referring to class *HeatExchangeType* (which has been dropped), all properties are now within the class (*internalHeatGainsConvectiveFraction*, *internalHeatGainsLatentFraction* and *internalHeatGainsRadiantFraction*). The original property *averageInternalGains* has been renamed *internalHeatGains*, is now of type Measure and is meant to contain the average value of the internal heat gains
- Class *Occupancy* has been simplified when it comes to property *heatDissipation*. Instead of referring to class *HeatExchangeType* (which has been dropped), all properties are now within the class (*heatDissipationConvectiveFraction*, *heatDissipationLatentFraction* and *heatDissipationRadiantFraction*). Property *heatDissipation* is now of type Measure and contains the average value of heat dissipation
- All *Facilities*-related classes have been moved, in simplified form, to the Devices module
- The *Household* class is dropped.

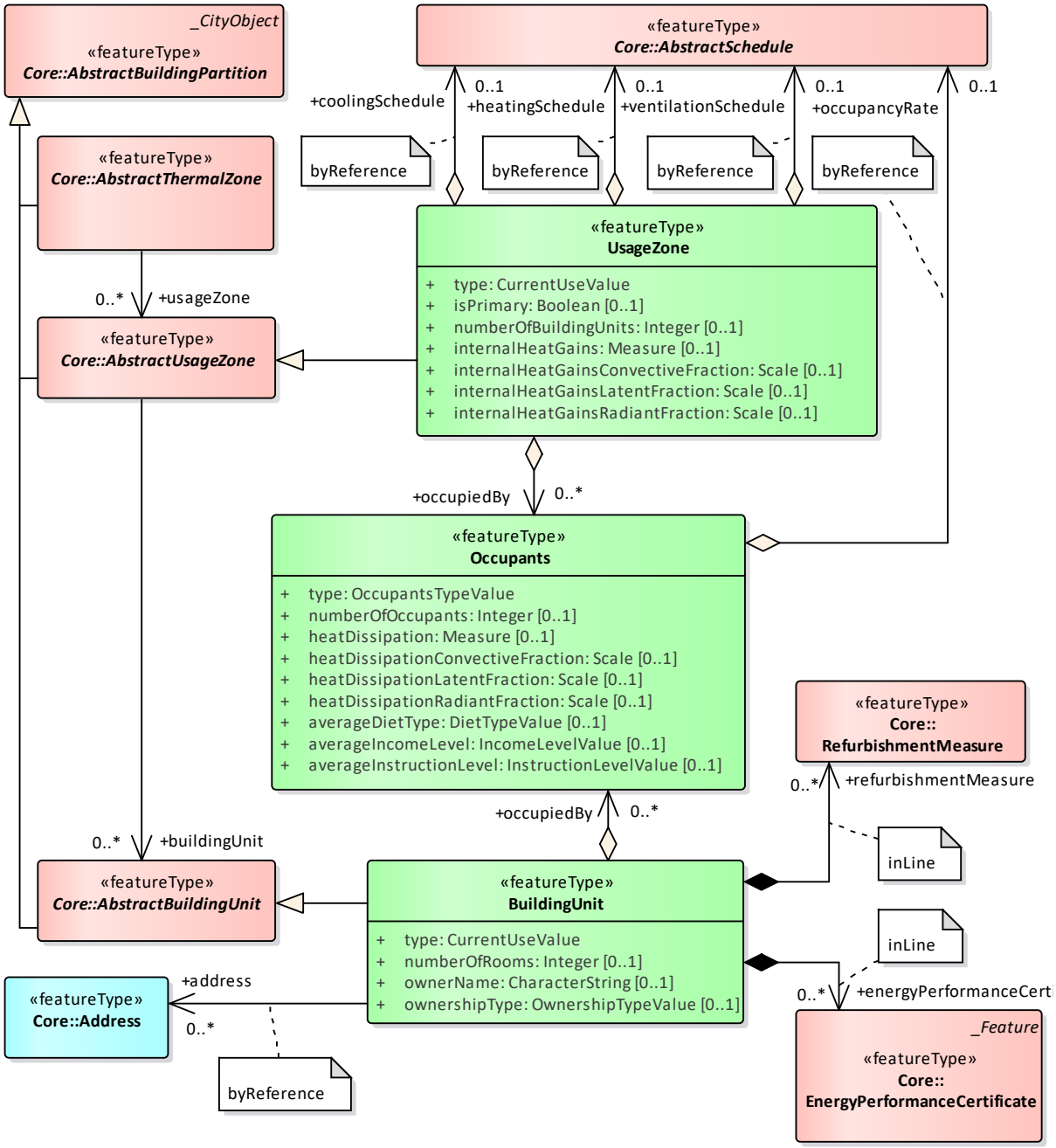


Figure 9. Overview of the Energy ADE 3.0 Occupancy module.

3.5 The Devices module

3.5.1 Characteristics

The Devices module represents the energy-relevant devices to perform building energy analyses. Its overview in terms of UML diagrams is presented in Figure 10.

The Devices module is related to Core module by means of class *AbstractDevice*. Any *_CityObject* class is associated with class *AbstractDevice* by means of a relation that has a multiplicity of 0 to many. Class *AbstractDevice* has several attributes that can be used to specify the characteristics of the device (or group of devices) via attributes such as *model*,

yearOfManufacture, *numberOfDevices*. More energy-related properties are provided via *installedPower*, *nominalEfficiency*, *efficiencyIndicator*, and *heatDissipation*. Finally, any *AbstractDevice* can be connected to *DeviceOperation* objects, which allows to determine – by means of schedules (see later) – how that device is used or operated.
From *AbstractDevice*, different specific classes are derived that represent devices, ranging from electrical and thermal energy storage (*ThermalStorageDevice* and *ElectricalStorageDevice*) to production and conversion (*HeatPump*, *Boiler*, *GenericElectricalDevice*, *LightingDevice*, *GenericDevice*). For some of these specific classes, some specific attributes are also available.
For solar collectors, for example, a specific subclass exists, called *AbstractSolarCollector*. This class possesses specific attributes regarding geometrical properties of the solar collectors (*moduleArea* and *apertureArea*, as well as *azimuth* and *inclination*). Additionally, two geometrical representations are also available, in LoD2 and LoD3 (*lod2MultiSurface* and *lod3MultiSurface*, respectively). From *AbstractSolarCollector*, four subclasses are derived, to be used depending on the type of solar collector: *SolarThermalCollector*, *PhotovoltaicCollector*, *PhotovoltaicThermalCollector*, and – in case no specific information is available – *GenericSolarCollector*.
For (movable) shading devices, class *MovableShadingDevice* can be used. Besides specifying the *type*, further attributes are the *installationSide* (i.e. inside or outside), the *maximumCoverRatio*, and the *transmittance*.

3.5.2 Main changes from the Energy ADE 1.0

The Devices module (formerly called Energy systems module) has undergone the most relevant changes since the Energy ADE 1.0. The overall structure has been sensibly reduced and simplified, also considering that the KIT Profile had it completely dropped. In short, the current module in the Energy ADE 3.0 represents somehow an “in-between” version, which has simplified the rather complex structure of the Energy ADE 1.0, but that still retains some of the basic classes.

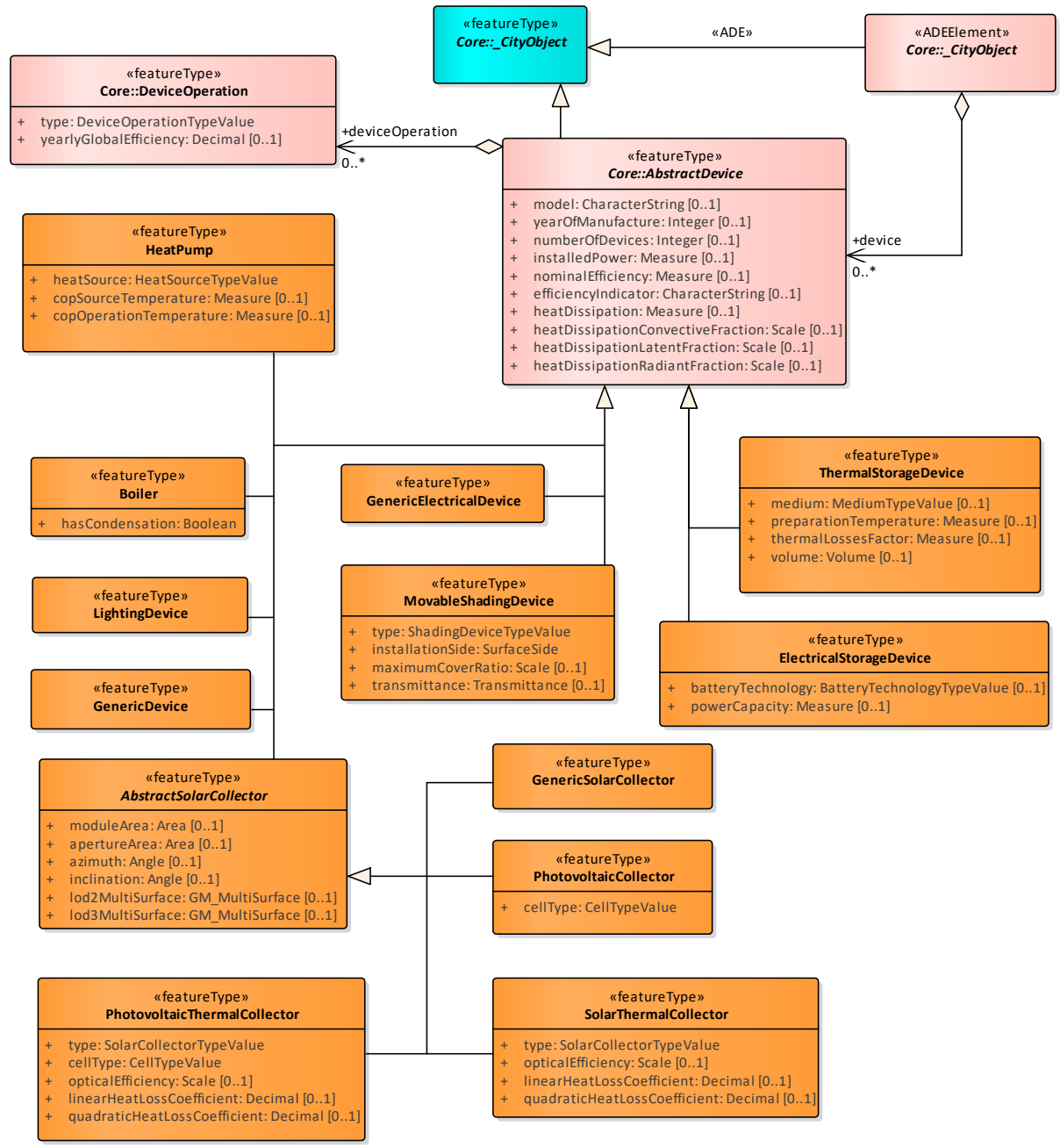


Figure 10. Overview of the Energy ADE 3.0 Devices module.

3.6 The Urban function areas module

3.6.1 Characteristics

The Urban function areas module is a new module added to the Energy ADE 3.0. It stems from the need to offer support for aggregated values (e.g. of energy demand, or of any resource – see the Resources module in section 3.6) at different geographical scales. Very often, values are computed and associated at building level, and must be aggregated and presented at different scales, e.g. at block, quarter, district level, or even up to the whole city. On the other hand, sometimes certain values are available only at such aggregated levels, therefore a geographical object is required to “contain” them. The overview of the urban function area module in terms of UML diagrams is presented in Figure 11.

The Urban function areas module takes inspiration from, and nearly coincides with, the *UrbanFunctionArea* class found in the i-UR ADE (Akatoshi et al., 2020), and it consists, basically, of just a class derived from CityGML class grp::*CityObjectGroup*. All properties of a *CityObjectGroup* class are inherited, i.e. attributes, as well as the possibility to associate a geometry (e.g. a polygon representing the administrative boundaries), the possibility to create hierarchies (*CityObjectGroups* containing *CityObjectGroups*), and the possibility to associate any city object to a group.
Class *UrbanFunctionArea* contains only few more properties. A property *type* associated to a CodeList (e.g. block, neighbourhood, district, city) and, optionally, a code to store specific existing codes associated with that *UrbanFunctionArea* (e.g. an ID used for a specific census area).
It is worth keeping in mind that the geometry associated to a *UrbanFunctionArea* object can be regular or irregular, e.g. a regular grid (as in the case of some existing datasets containing statistical data in some countries) or an administrative boundary, or any other user-defined shape.

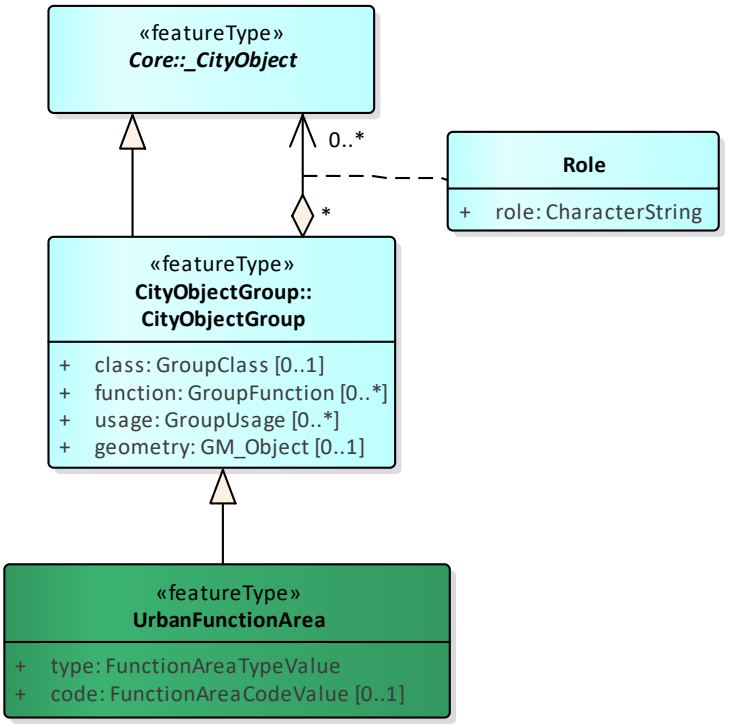


Figure 11. Overview of the Energy ADE 3.0 Urban function areas module.

3.7 The Weather station module

3.7.1 Characteristics

The Weather station module contains just one class, *WeatherStation*. This class inherits from the Core module the possibility to be related to zero or several *WeatherData* objects and is the only member of this module. The overview of the Weather station module in terms of UML diagrams is presented in Figure 12.

3.7.2 Main changes from the Energy ADE 1.0

Compared to the Energy ADE 1.0, there are no major differences, except for two modelling errors that have been corrected. They are:

- The Energy ADE 1.0 *WeatherStation* property *stationName* has been dropped, as already inherited from *_Feature* (via the property *name*)
- The Energy ADE 1.0 *WeatherStation* property *weatherData* has been dropped, as it is already inherited from *_CityObject*.

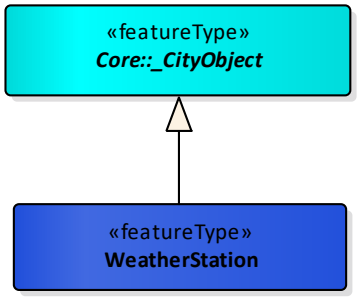


Figure 12. Overview of the Energy ADE 3.0 Weather station module.

3.8 The Layered construction module

3.8.1 Characteristics

The Layered construction module physically characterises how the building construction parts are made. It allows to detail their structure and to specify their thermal and optical properties. The overview of the Layered construction module in terms of UML diagrams is presented in Figure 13.
The Layered construction module is related to the Core module by means of class *AbstractLayeredConstruction*. Two classes are derived from it:

- The *LayeredConstruction* class, which contains information about the thermal and optical properties of the layered construction as a whole, and – optionally – the set of layers composing it. Layers are ordered from the inside of a room towards the outside.
- The *ReverseLayeredConstruction* class, which is used to simply provide a way to represent layered constructions for which the order of the layers has to be inverted. This could be the case, for example, when associating a construction to party wall surfaces.

A *LayeredConstruction* class contains information about the overall thermal transmittance of the whole construction (property *uValue*), intended as a set of layers having each certain physical properties. Additionally, optical properties such as transmittance, reflectance emissivity and (if applicable) the glazing ratio, can be specified. The *gValue* attribute corresponds to the coefficient used to measure the transmittance of solar gain through glazing, considering also the frame. This is a value that is generally available from the windows manufacturers of windows. However, it is possible to model this value also by means of the glazingRatio and the optical properties values (e.g. transmittagce), in case more detailed data are available.

The *libraryCode* attribute can be used to store a specific code (with the associated codespace) for a specific layered construction to be looked up in an external database.
A *LayeredConstruction* object can be used as such, or can be further detailed by specifying an ordered set of *Layer* objects, each one having its own *thickness* and being related to a specific material (be it a *Gas* or a *SolidMaterial*, through their common parent class *AbstractMaterial*). Conventionally, the order of the layers follows the direction of the normal vector of the surface they refer to. A *libraryCode* attribute can be used to store a specific code (with the associated codespace) for a specific *AbstractMaterial* to be looked up in an external database.
If the order of the layers is opposite to the intended direction, then class *ReverseLayeredConstruction* can be used. It contains a relation to the original *LayeredConstruction* object, and it is meant to indicate the inversion of the layers order.
Each subclass of *AbstractMaterial* has some specific attributes.

Two more classes are contained in Figure 13: the *MaterialLibrary* and the *LayeredConstructionLibrary*.
The *MaterialLibrary* class is derived from *AstractLibrary* in the Core module and is intended to contain *all* instances of the *Gas* or *SolidMaterial* classes. As indicated in Figure 13, all *Layer* objects will refer to the corresponding material object by mean of a reference (e.g. an XLink), as material are considered to be entities that can be re-used many times.
The same concept applies to class *LayeredConstructionLibrary*. As it is the case with construction information from external libraries (e.g. TABULA), *all* LayeredConstruction objects belonging to the same library must be contained only within the same *LayeredConstructionLibrary* object. All city objects referring to a layered construction (be it a *LayeredConstruction* or a *ReverseLayeredConstruction* object) will refer to it by a reference.
Although in XML the concept of resources re-use and linkage is already available through the XLink mechanism, a *MaterialLibrary* or a *LayeredConstructionLibrary* are meant to be the *only* place where such instance objects are to be found, as this modelling technique is more in line with the “logic” followed by energy simulation tools. More details and modelling examples are provided in the Modelling rules (section 4.8).
The advantage of collecting all (*Reverse*)*LayeredConstruction* objects within one library is that, among the rest, the library itself can be exchanged as a single object. As a matter of fact, it is derived from *_CityObject* class and is intended to be a top-level class.

3.8.2 Main changes from the Energy ADE 1.0

This module essentially corresponds to the Construction module of the Energy ADE 1.0, with relatively few changes. However, its name has been changed as CityGML 3.0 has introduced an homonymous module Construction and an homonymous class *Construction* that have however a different semantic meaning and different scope. Therefore, to avoid naming issues for the future mapping of the Energy ADE 3.0 to CityGML 3.0, the module and the main class have been renamed. The other differences compared to the Energy 1.0 are:

- Class *LayeredConstruction* has an additional attribute *libraryCode*, meant to identify a specific object with its code used possibly in an external library. Additionally, class *LayeredConstruction* has been simplified when it comes to the optical properties. Instead of referring to class *OpticalProperties* (which has been dropped), all properties are now within the class (*glazingRatio*, *emissivity*, *reflectance*, *transmittance*)

- Similarly to class *LayeredConstruction*, class *AbstractMaterial* has an additional attribute *libraryCode*
- Class *LayerComponent* has been dropped. Property *thickness* has been moved to class *Layer*, which is now directly connected to *AbstractMaterial*
- Class *ServiceLife* has been removed
- Class *ImageTexture* has been removed
- The concept of libraries, both for materials and layered constructions, is new.

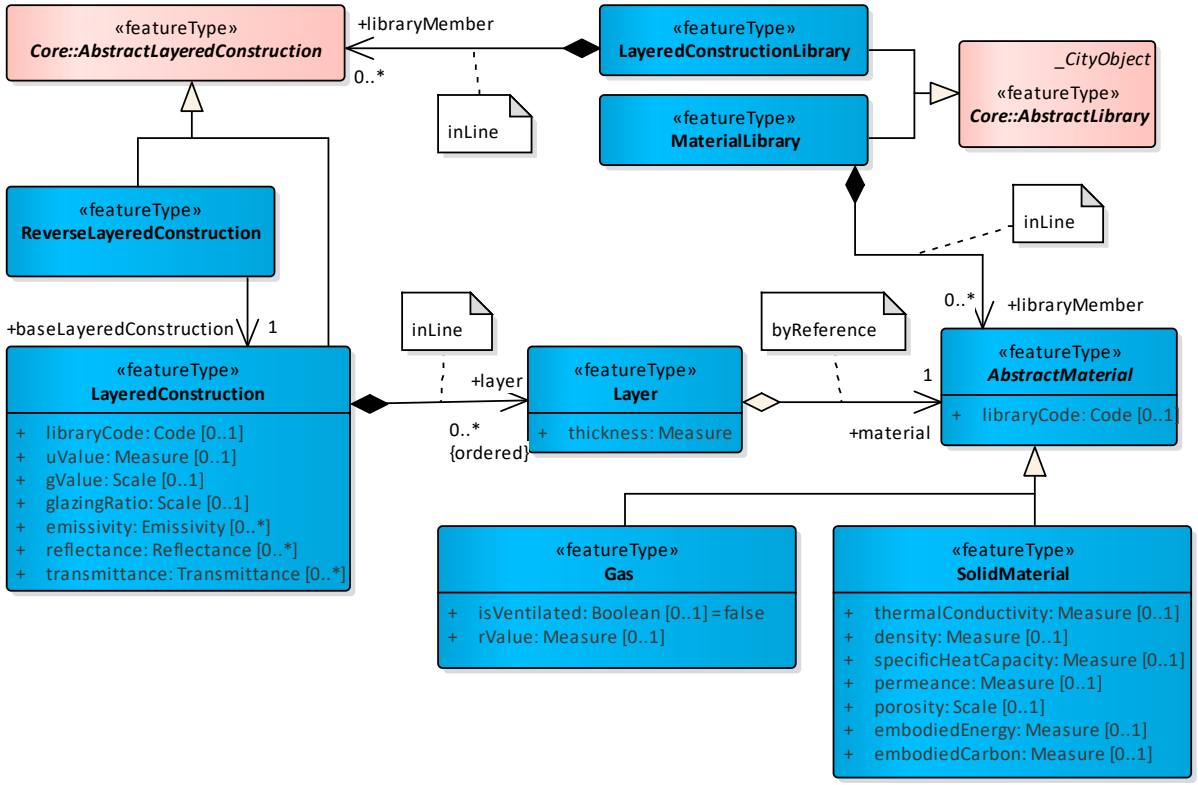


Figure 13. Overview of the Energy ADE 3.0 layered construction module.

3.9 The Resources module

3.9.1 Characteristics

The Resources module is a new module added to the Energy ADE 3.0. It stems from the need to extend and generalise the concept of *EnergyDemand* in the Energy ADE 1.0, in which a *_CityObject* could only be related to different types of *EnergyDemand* (for heating, cooling, cooking, etc.). However, neither the production of heat/electricity (e.g. by means of photovoltaic solar collectors or from geothermal energy sources), nor potential values of production, or demands (or storage) could be explicitly modelled. Finally, the demand was limited to energy, but any other types of resources were left out (e.g. water, food, etc.).
The resource module takes inspiration from the Food-Water-Energy ADE (Padsala et al., 2021), and largely extends the modelling capabilities of the Energy ADE, allowing to overcome the aforementioned shortcomings of the Energy ADE 1.0. The overview of the resources module in terms of UML diagrams is presented in Figure 14.
The main idea is that *every* city object can demand/consume, produce/generate, store/accumulate (this is expressed by attribute *operationType*) "something" (i.e. a resource).

Thanks to the attribute *status*, it is now possible to deal not only with actual values, but also with *potential* values.
The quantity can be expressed either as a yearly amount, or as a time series (or both), via the attribute *amount* and the *timeDependentAmount* property, respectively. Values can be expressed either as absolute or specific values providing additional information via the attributes *isAmountNormalized*, *narmalizationValue* and *normalizationParameter*. For example, value(s) of energy demand could be expressed in in kWh/a. However, if the normalization value is provided (e.g. heated reference area in m^2) then the value(s) in kWh/(a*m^2) can be obtained from the previous without the need to store two values or two time series. Finally, the CO_2-equivalent amount for each resource can be specified, as well as its costs for production or consumption.
Class *AbstractResource* is finally specialized in several subclasses, representing each a specific type of resource. i.e. *Energy*, *Water*, *Food*, *Waste*, *ConstructionMaterial*, *UrbanSpace*, or a generic *OtherResource*. For each of these subclasses, specific attributes are available too.
Finally, it is worth mentioning that the resources module may offer new possibilities to deal with circularity of materials at urban scale, as it allows to associate to each city object, which acts as a node in a graph, it owns quantities of resources produced, consumed or stored. By using the *CityObjectRelation* class to establish the link between nodes, a graph with production and consumption nodes for different types of resources could be created.

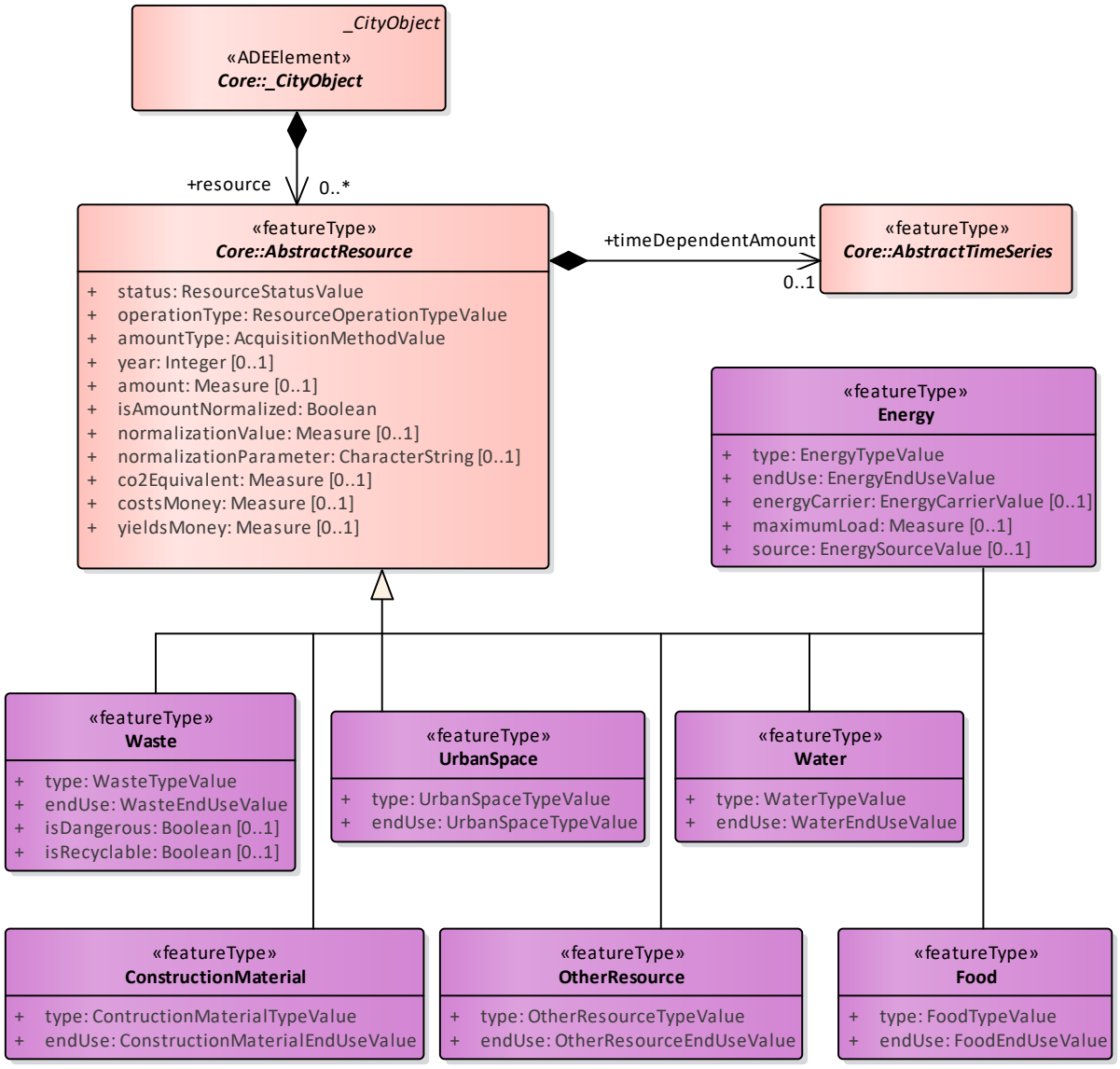


Figure 14. Overview of the Energy ADE 3.0 Resources module.

3.10 The Schedules module

3.10.1 Characteristics

The Schedules module contains classes that are needed to define schedules. The UML diagram is presented in Figure 15. This module has changed considerably compared to the Energy ADE 1.0. Its current design is more in line with the Dynamizer module of CityGML, as it follows a rather similar logic despite some minor differences. Still, a future mapping of the Schedules module to CityGML 3.0 will be relatively straightforward.

All schedules are derived from the abstract base class *AbstractSchedule*. The *AbstractSchedule* class contains optional properties (*startTime*, *startDay*, *startMonth*, *startYear*) that can be used to specify the precise timestamp when the schedule begins (e.g. at 00:00:00 of 1 January 2025). Alternatively, the begin time can be expressed also in relative terms (e.g. 00:00:00 of 1 January), meaning that such schedule may start *any* 1 January at 00:00:00. The property *temporalExtent* can be used to define the total extent of a schedule (1 day, 1 year, etc.). Property *type* provides information about the type of the schedule, e.g. a daily schedule, a

typical-day schedule, a schedule happening on Monday, etc. Finally, property *libraryCode* can be used to associate a code to the schedule that can be further looked up in an external database.
There the specific types of schedules. The *DualValueSchedule* is meant to model simple schedules with an *idleValue*, and a *usageValue*, the latter being valid between a *startUsageTime* and a *endUsageTime* during the day.
An *AtomicSchedule* is a schedule which is used to model either a constant-value schedule (e.g. a set point) via its *constantValue* property, or that can be connected to an *AbstractTimeSeries*. Again, properties inherited from *AbstractTimeSeries* are used to define the *type* of schedule, and to optionally characterize its beginning point in time, and its overall duration. Examples of atomic schedule objects are schedules of typical days, associated to time series containing hourly values (e.g. occupancy curves, or similar). The specific values, their time resolution, etc. are then contained in the time series associated with it.
AtomicSchedule (but also *DualValueSchedule* and *CompositeSchedule*) objects can be composed into a *CompositeSchedule* object. In this case, additional information about each component schedule is provided by class *ScheduleComponent*. This specifies the *type*, the *repetitions* number (if applicable) and optionally, the *additionalGap* between repetitions. For example, using a *CompositeSchedule* object, it is possible first to define two *AtomicSchedule* objects, e.g. one with hourly values for a typical week day and one with hourly values for a typical weekend day. The typical weekday schedule is then linked by a *ScheduleComponent* object (e.g. with 5 as repetition value), while the other schedule for the typical weekend day is linked by another *ScheduleComponent* object (e.g. with repetition value of 2). The resulting *CompositeSchedule* object would be composed by the two *ScheduleComponent* objects, covering a typical week of 7 days: 5 times a typical week day, followed by 2 times a typical weekend day. Similar compositions can be of course extended to different periods, spanning over longer periods (e.g. years composed of seasons or months, etc.).

One more class is contained in Figure 15: the *ScheduleLibrary*. The *ScheduleLibrary* class is new and is derived from *AbstractLibrary* in the Core module. It is intended to contain *all* schedule objects. All Energy ADE 3.0 classes connected to a schedule object will refer to it by means of reference (e.g. an XLink), as schedules are typical objects that can be re-used many times.
Although in XML the concept of resources re-use and linkage is already available through the XLink mechanism, a *ScheduleLibrary* is meant to be the *only* place where a set of schedules is to be found, as this modelling technique is more in line with the “logic” followed by energy simulation tools. More details and modelling examples are provided in the Modelling rules (section 4.10).
The advantage of collecting all schedule objects within one library is that, among the rest, the library itself can be exchanged as a single object. As a matter of fact, it is derived from *_CityObject* class and is intended to be a top-level class.

3.10.2 Main changes from the Energy ADE 1.0

The major changes compared to the Energy ADE 1.0 are listed as follows:

- The stereotypes of the classes has been changed from «type» to «featureType»
- The *AbstractSchedule* has an additional attribute *libraryCode*, meant to identify a specific object with its code used possibly in an external library

- Classes *ConstantValueSchedule*, *DailyPatternSchedule* and *TimeSeriesSchedule* have been dropped. As a consequence, also classes *PeriodOfYear* and *DailySchedule* have been dropped
- Classes *ConstantValueSchedule, DailyPatternSchedule* and *TimeSeriesSchedule* can now be expressed by means of the new classes *AtomicSchedule* and *CompositeSchedule*
- The concept of a library containing all schedules is new.

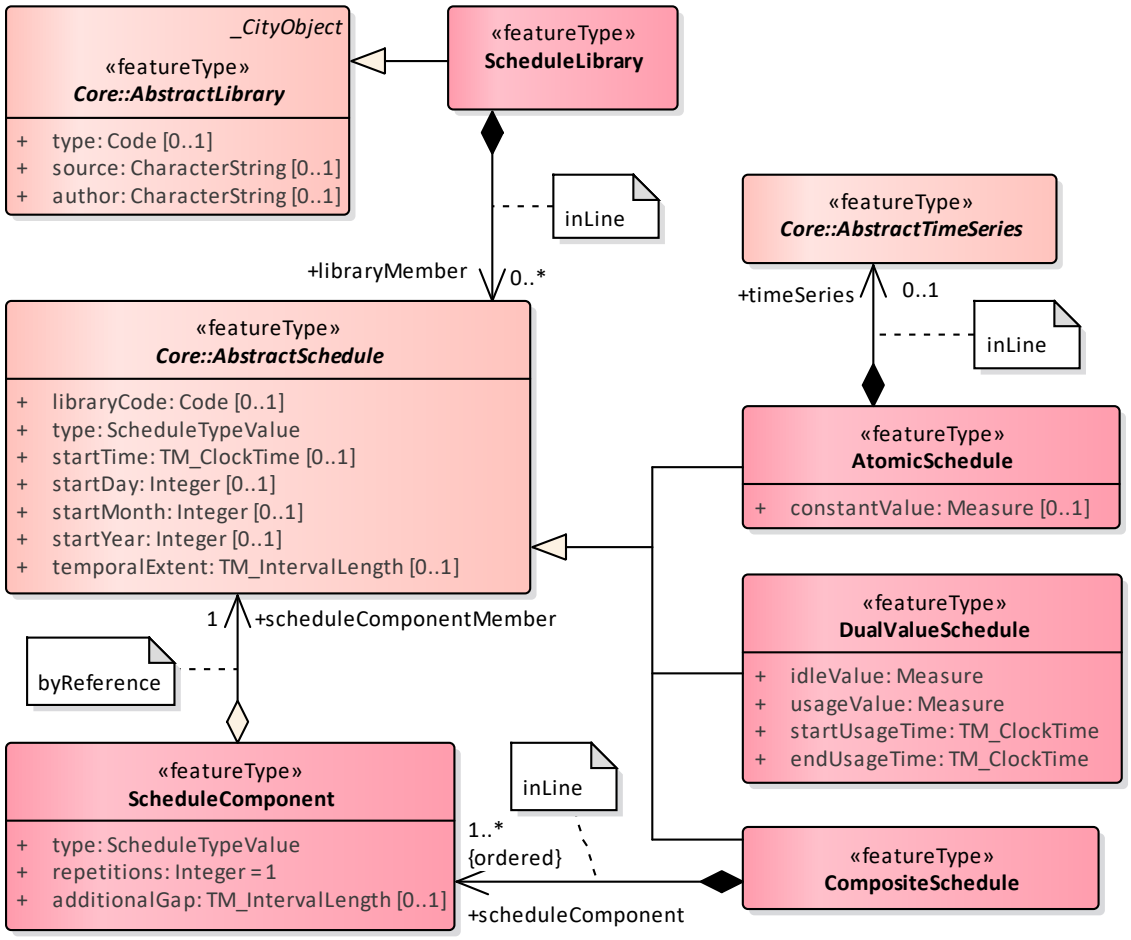


Figure 15. Overview of the Energy ADE 3.0 Schedule module.

3.11 The Time series module

3.11.1 Characteristics

The modules previously described represent time-dependent properties by means of the class *AbstractTimeSeries* and its derived classes. The overview of the time series classes in terms of UML diagrams is presented in Figure 16. All specific time series are derived from *AbstractTimeSeries* and inherit its properties *acquisitionMethod*, *interpolationType* and *source*, which allow to provide some metadata information about the time series.

For regular time series, the values have a defined start and end time (*temporalExtent*) using timestamps related to absolute time, and a constant time increment (*timeInterval*). The time series values themselves may either be stored directly in the same document (class *RegularTimeSeries*) or in a separate file with table structure (class *RegularTimeSeriesFile*). In

the latter case, additional properties are provided to further describe the structure of the file (e.g. a csv file).
A time series containing typical values, i.e. not related to a specific point in absolute time, can be modelled defining the starting point (*startTime*, *startDay*, *startMonth*), its temporal extent (*temporalExtent*), and a constant time increment (*timeInterval*). Again, the time series values themselves may either be stored directly in the same document (class *TypicalValuesTimeSeries*) or in a separate file with table structure (class *TypicalValuesTimeSeriesFile*). In the latter case, additional properties are provided to further describe the structure of the file.
Finally, class *SensorConnection* is backported from CityGML 3.0 and can be used to store metadata required to connect to a sensor.

3.11.2 Main changes from the Energy ADE 1.0

The most relevant changes compared to the Energy ADE 1.0 are:

- The stereotypes of the classes has been changed from «type» to «featureType»
- In class *AbstractTimeSeries* the set of metadata attributes has been simplified
- Classes *TypicalValuesTimeSeries*, *TypicalValuesTimeSeriesFile* and *SensorConnection* have been added
- Like in the Energy ADE 1.0 KIT profile, all classes related to *IrregularTimeSeries* have been dropped.

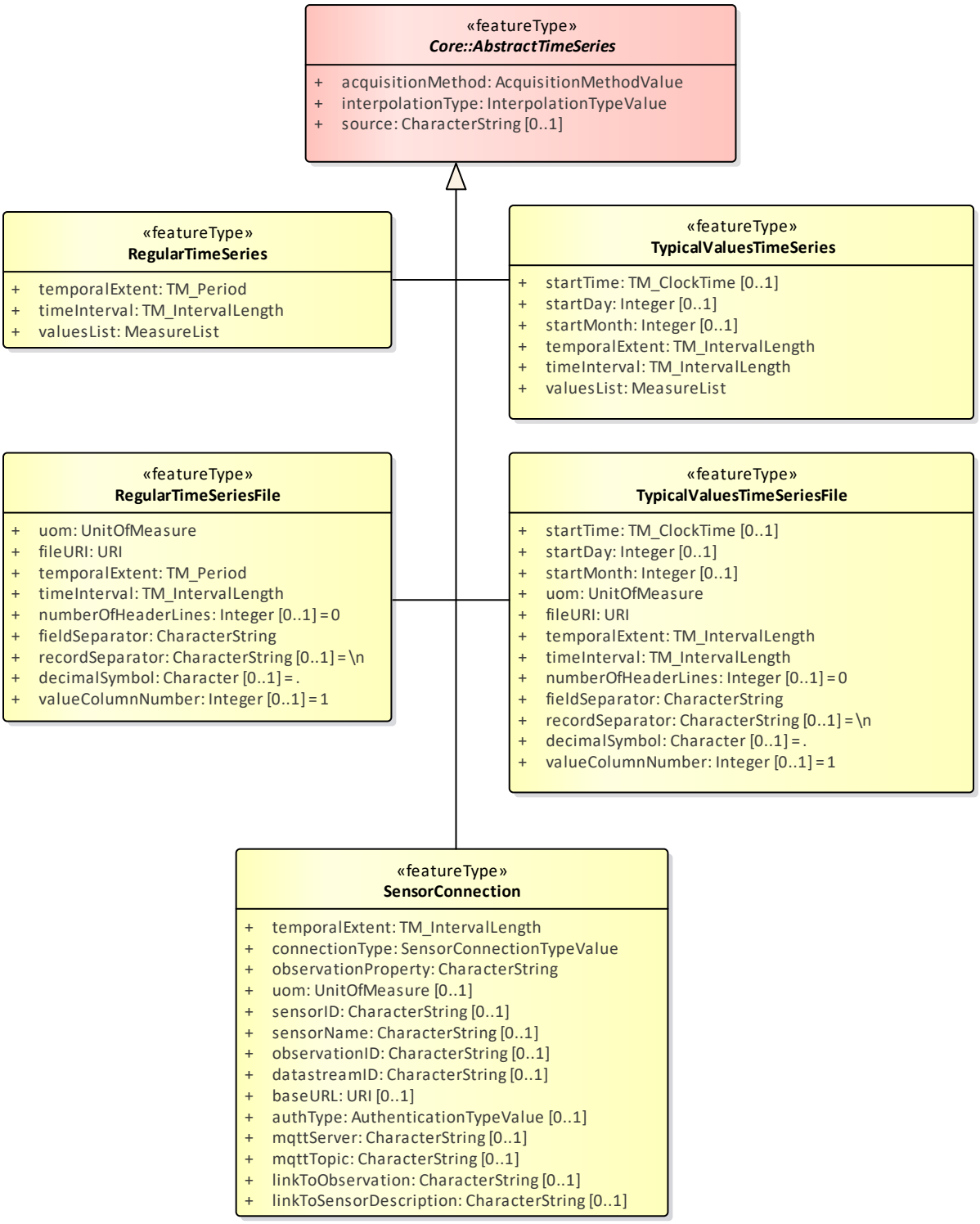


Figure 16. Overview of the Energy ADE 3.0 Time series module.

4 Modelling rules and guidelines

This section contains some modelling rules and guidelines that can help in the process of creating and writing datasets that contain data in CityGML 2.0 with Energy ADE 3.0 contents. These modelling rules and guidelines refer to CityGML documents encoded in XML.

4.1 General rules

4.1.1 Rule 1: "InLine" or "byReference" elements

In order to avoid too many different modelling possibilities, the UML diagrams provide annotations indicating whether the corresponding target elements must be written in the XML document as *in-line* elements or *referenced* by means of XLinks when it comes to their parent/containing element. If no annotations are provided, then the default behaviour is in line *or* by reference, i.e. *both* possibilities are allowed.

An example is provided in Figure 17 using an excerpt of XML which contains elements from classes *Building*, *Address*, *BuildingUnit*, *UsageZone*, *ThermalZone, Occupants* (please refer to Figure 7 for the corresponding UML diagram). It can be observed that the building element contains elements from classes *Address*, *BuildingUnit*, *UsageZone*, and *ThermalZone*. Every element is written *in-line*, i.e. they are written as direct children of the *Building* element. The different indentation of the lines helps recognise the hierarchy. The same applies to the *Occupants* element with respect to the containing *UsageZone* element. These are examples of in-line elements.

When it comes to referencing elements, it can be seen in Figure 17 that the *BuildingUnit* element is referenced from within the containing *UsageZone* element, using an XLink (line 31) that contains the id of the referenced element ("id_building_unit_1") with the prefix "#" – as required when linking a resource in the same document. Similarly, the *UsageZone* element is referenced from within the containing *ThermalZone* element, using another XLink (line 20), i.e. "#id_usage_zone_1".
Finally, the same mechanism is visible also in line 12, where the *BuildingUnit* element references the *Address* element with id "id_address_1". By the way, this last example shows how **Rule 5** (see section 4.4.1) must be applied.

```
<core:cityObjectMember>
    <bldg:Building gml:id="id_building_1">
        <gml:description>This is Building 1</gml:description>
        <gml:name>Snoke's Palace</gml:name>
        ...
        <bldg:address>
            <core:Address gml:id="id_address_1">
            ...
            </core:Address>
        </bldg:address>
        ...
        <nrg3:buildingUnit>
            <nrg3:BuildingUnit gml:id="id_building_unit_1">
                <gml:description>This is BuildingUnit 1</gml:description>
                <gml:name>BuildingUnit 1</gml:name>
                ...
                <nrg3:type codeSpace="buildingUnit_codeSpace_123">residential</nrg3:type>
                <nrg3:address xlink:href="#id_address_1"/>
            </nrg3:BuildingUnit>
        </nrg3:buildingUnit>
        <nrg3:thermalZone>
            <nrg3:ThermalZone gml:id="id_thermal_zone_1">
                <gml:description>This is ThermalZone 1</gml:description>
                <gml:name>ThermalZone 1</gml:name>
                ...
                <nrg3:usageZone xlink:href="#id_usage_zone_1"/>
                ...
                <nrg3:coincidesWithLod2Hull>false</nrg3:coincidesWithLod2Hull>
                <nrg3:coincidesWithLod3Hull>false</nrg3:coincidesWithLod3Hull>
            </nrg3:ThermalZone>
        </nrg3:thermalZone>
        <nrg3:usageZone>
            <nrg3:UsageZone gml:id="id_usage_zone_1">
                <gml:description>This is UsageZone 1</gml:description>
                <gml:name>UsageZone 1</gml:name>
                ...
                <nrg3:buildingUnit xlink:href="#id_building_unit_1"/>
                <nrg3:type codeSpace="usageZone_codeSpace_123">residential</nrg3:type>
                ...
                <nrg3:occupiedBy>
                    <nrg3:Occupants gml:id="id_occupants_1">
                        <gml:description>This is Occupants 1</gml:description>
                        <gml:name>Occupants 1</gml:name>
                        ...
                    </nrg3:Occupants>
                </nrg3:occupiedBy>
            </nrg3:UsageZone>
        </nrg3:usageZone>
    </bldg:Building>
</core:cityObjectMember>
```

Figure 17. Excerpt of XML showing examples of elements modelled inline and by reference (i.e. using XLinks).

4.1.2 Rule 2: Geometries and XLinks

Just like in CityGML 2.0, XLinks can be used by one feature to reference geometries (more specifically: polygons) so that such geometries can be reused several times within the same instance document without having to explicitly rewrite the coordinates, but simply referring to the *gmlid* of said geometry. Compared to the CityGML 2.0 specifications, and in line with CityGML 3.0, the Energy ADE 3.0 introduces an additional constraint when it comes to using XLinks for geometries, which is:

- XLinks are allowed only within geometries belonging to the same city object (e.g. the same building) and within the same LoD.

For example: Geometries composing the LoD2 thermal boundaries of a LoD2 thermal zone can be referenced by XLinks to define the LoD2 solid geometries of that thermal zone, but not for the LoD3. Vice versa, if applicable, LoD3 thematic surfaces or openings of a building can be

used to create LoD3 thermal boundaries and thermal openings, but only for the same building and the same LoD3.

4.2 Core module

4.2.1 Rule 3: Writing *CityObjectRelation* elements

The UML association class *CityObjectRelation* is modelled as an intermediate class before the GML encoding is applied. In this way, a relation between two city objects, e.g. city object "A" and city object "B" can be represented in such a way that city object A provides the element *CityObjectRelation* inline, whereas the *CityObjectRelation* references city object B using an XLink. This modelling rule is in accordance with the analogous Requirement 6 of CityGML 3.0[14]. Figure 18 provides a UML-based representation of the concept, while an example in terms of XML encoding is provided in Figure 19.

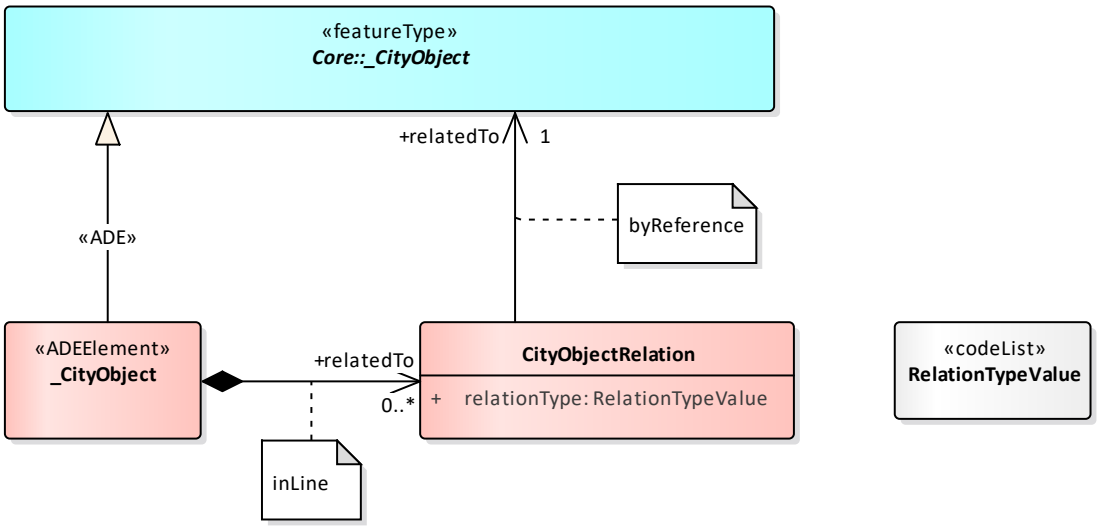


Figure 18. UML diagram of the intermediate modelling of association class "CityObjectRelation" before the GML encoding is applied.

[14] https://docs.ogc.org/is/21-006r2/21-006r2.html#req_global_alternativeaggregations

```
<core:cityObjectMember>
    <bldg:Building gml:id="id_building_1">
        <gml:description>This is Building 1</gml:description>
        <gml:name>Snoke's Palace</gml:name>
        ...
        <nrg3:relatedTo>
            <nrg3:CityObjectRelation>
                <nrg3:relationType codeSpace="adjacency_codeList">adjacent</nrg3:relationType>
                <nrg3:relatedTo xlink:href="#id_building_2"/>
            </nrg3:CityObjectRelation>
        </nrg3:relatedTo>
        ...
    </bldg:Building>
</core:cityObjectMember>
<core:cityObjectMember>
    <bldg:Building gml:id="id_building_2">
        <gml:description>This is Building 2</gml:description>
        <gml:name>Rey's Hut</gml:name>
        ...
        <nrg3:relatedTo>
            <nrg3:CityObjectRelation>
                <nrg3:relationType codeSpace="adjacency_codeList">adjacent</nrg3:relationType>
                <nrg3:relatedTo xlink:href="#id_building_1"/>
            </nrg3:CityObjectRelation>
        </nrg3:relatedTo>
        ...
    </bldg:Building>
</core:cityObjectMember>
```

Figure 19. Excerpt of XML showing examples of city objects (here: two buildings) making use of the *CityObjectRelation* association to model their mutual adjacency.

4.2.2 Rule 4: Hierarchy of *ThermalZone*, *UsageZones* and *BuildingUnit* elements

In order to avoid too many different possibilities to write elements representing these classes, and their mutual relations, the following rule applies:

- All *ThermalZones*, *UsageZones*, and *BuildingUnits* elements are written as direct children of the containing element derived from ADE *_AbstractBuilding*, (i.e. *Building* or *BuildingPart*)
- The relations between *ThermalZones* and *UsageZones*, and between *UsageZones* and *BuildingUnits* are expressed only by means of XLinks. Therefore, no *BuildingUnit* element must be written as an inline element of a *UsageZone* element, and no *UsageZone* element must be written as inline element of a *ThermalZone* element

The UML diagrams provide some annotations with regard to this rule. An example is provided in Figure 17.

4.2.3 Writing *UtilityNetworkConnection* elements

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, elements of classes *EnergyPerformanceCertificate* and *RefurbishmentMeasure* must be written inline as children of the respective containing city object element.

4.2.4 Writing *WeatherData* elements

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, *WeatherData* elements must be written inline as children of the respective containing city object element.

4.2.5 Writing *EnergyPerformanceCertificate* and *RefurbishmentMeasure* elements

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, *EnergyPerformanceCertificate* and *RefurbishmentMeasure* elements must be written inline as children of the respective containing element (*Building* or *BuildingPart*).

4.2.6 Writing resource elements

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, resource elements corresponding to any subclass of *Abstractresource* must be written inline as children of the respective containing city object element.

4.2.7 Writing the association from ADE *_CityObject* to *AbstractLayeredConstruction*

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, the association between ADE *_CityObject* and *AbstractLayeredConstruction* is of type “byReference”. Therefore, it must be written using an XLink. An example is provided in Figure 20.

```
<core:cityObjectMember>
    <bldg:Building gml:id="id_building_1">
        <gml:description>This is Building 1</gml:description>
        <gml:name>Snoke's Palace</gml:name>
        ...
        <bldg:boundedBy>
            <bldg:RoofSurface gml:id="id_building_1_roofsurface_1">
                ...
                <nrg3:layeredConstruction xlink:href="#id_layered_construction_roof_3"/>
                ...
            </bldg:RoofSurface>
        </bldg:boundedBy>
        <bldg:boundedBy>
            <bldg:WallSurface gml:id="id_building_1_wallsurface_1">
                ...
                <nrg3:layeredConstruction xlink:href="#id_layered_construction_wall_2"/>
                ...
            </bldg:WallSurface>
        </bldg:boundedBy>
    </bldg:Building>
</core:cityObjectMember>
```

Figure 20. Excerpt of XML showing examples of city object elements (here: from *RoofSurface* and *WallSurface*) referring to layered construction elements by means of XLinks.

4.2.8 Writing the association from *DeviceOperation* to *AbstractSchedule*

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, the association between *DeviceOperation* and *AbstractSchedule* is of type “byReference”, therefore it must be written using an XLink. Please refer to section 4.10 for more details on how to model and write schedules.

4.2.9 Writing the association to *AbstractTimeSeries*

Please refer to section 4.11 for more details on how to model and write time series elements.

4.3 Building physics module

4.3.1 Modelling *ThermalZone* objects

The following points provide further details regarding common cases that may occur when modelling a *ThermalZone*, depending on the LoD and on whether or not thermal boundaries are used. More specifically:

- In LoD0, no geometrical representation of *ThermalZones* is allowed
- In LoD1, *ThermalZones* can be modelled via a solid geometry (inherited from core:*AbstractBuildingPartition*). However, no further modelling is allowed using thematic surfaces (in order to align with the standard CityGML rules)
- In LoD2, *ThermalZones* can be modelled using a LoD2 solid geometry (inherited from core:*AbstractBuildingPartition*) and/or thermal boundaries. No explicit modelling of thermal openings is allowed (as in CityGML), therefore geometries must be modelled without "holes" representing the openings. The property *totalSurfaceArea* attribute refers to the whole surface area, including both opaque and opening areas. Optionally, the *opaqueSurfaceArea* or the *openingToSurfaceRatio*[15] attributes can be used. If applicable. The former can be used to specify explicitly the amount of opaque area; the latter can be used to derive the portion of opaque and opening areas starting from the *totalSurfaceArea* value
- In LoD3, *ThermalZones* can be modelled using a LoD3 solid geometry (inherited from core:*AbstractBuildingPartition*) and/or thematic surfaces. In case openings exist, the geometry of the corresponding thematic surface must contain the "hole(s)". The *totalSurfaceArea* and the *opaqueSurfaceArea* attributes are used as in the case of LoD2: the former indicates the total area of the surface (including the openings), while the latter indicates only the area of the opaque surface
- For the opening(s), the corresponding CityGML classes (*Window* or *Door*) must be used. The geometry must be such that it fills the "hole" of the parent thematic surface. Either LoD3 multisurface or implicit-representation geometries can be used. The area attribute of the opening corresponds to the area of the geometry
- For LoD2, a special case applies. In the specific case that the thermal zone coincides *exactly* with the external hull resulting from the union of all LoD2 thematic surfaces, then the attribute *coincidesWithLoD2Hull* must be set to True and no further modelling via thematic surfaces is required. In this way, no duplicate objects must be created as the "normal" thematic surfaces can be used directly in lieu of thermal boundaries. This special case applies in the case of buildings modelled using a single thermal zone
- A second special case applies with LoD3. It is conceptually similar to the previous one: the attribute *coincidesWithLoD3Hull* must be set to True and no further modelling via thematic surfaces is required. Again, this special case applies in the case of buildings modelled using a single thermal zone
- Finally, *ThermalZones* are not allowed to be modelled in LoD4. Neither thermal boundaries nor thermal openings are allowed to use LoD4 geometries inherited from the respective CityGML classes

[15] The name is a more general form of the well-known *window-to-wall ratio* parameter.

- When modelling thermal openings in LoD3, particular care must be taken, as their geometries might differ from the respective geometries of the “normal” openings. Geometries of thermal openings must perfectly fill out the hole in the corresponding thermal boundary. This is not the case of the thematic surfaces and openings. It might in fact happen that the geometry of the opening is not coplanar with the thematic surface (e.g. a wall) it belongs to. Such an example if provided, schematically, in Figure 21.

An overview of such rules is provided in Table 1.

Table 1. Overview of modelling rules for thermal zones, thermal boundaries and thermal openings.

LoD	Semantics	Geometries
LoD1	• Thermal zones must be modelled • No thermal boundaries, no thermal openings • Optionally, *LayeredConstruction* objects associated to thermal zones	• Thermal zones: (optional) LoD1 solid geometries • No geometries for thermal boundaries / thermal openings
LoD2 & LoD3	• Thermal zones must be modelled • Thermal boundaries must be modelled via thematic surfaces • For LoD3: Thermal opening must be modelled (if they exist) via openings • Optionally, *LayeredConstruction* objects associated to thermal boundaries and thermal openings • Optionally, use *RelationToCityObject* to relate to corresponding thematic surface or opening	• Thermal zones: (optional) LoD2/3 solid geometries • Thermal boundaries: geometries must be modelled explicitly. In certain cases, XLinks to corresponding geometry of thematic surfaces are allowed, if in the same LoD and referring to the same Building(Part) • Thermal openings: geometries can be modelled explicitly or as implicit-representation geometries. In certain cases, XLinks to corresponding geometry of openings are allowed, if in the same LoD and in the same Building(Part) • Geometries of LoD3 thermal openings must perfectly fill the “hole” in the corresponding thermal boundary
LoD2 & LoD3 special cases	• Thermal zones must be modelled. Property *coincidesWithLoDxHull* must be set to TRUE • Instead of thermal boundaries / thermal openings, directly use thematic surfaces and openings • Optionally, *LayeredConstruction* objects associated to thematic surfaces and openings	• Thermal zones: (optional) LoD2/3 solid geometries • No geometries for thermal boundaries / thermal openings

Some schematic examples of what the rules mean in practice are provided in the Table 2. The following images use the following colour coding (an example is provided in Figure 21):

- **Black**: Building geometries (in different LoDs)
- **Cyan**: Geometries used for building openings
- **Orange**: Geometries used for thermal zones or for thermal boundaries
- **Dark cyan**: Geometries used for thermal openings

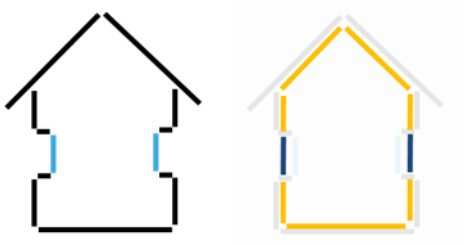
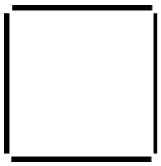
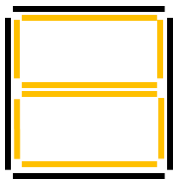
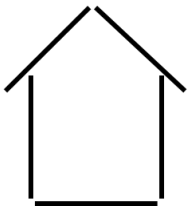
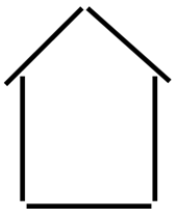
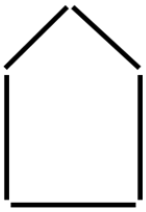
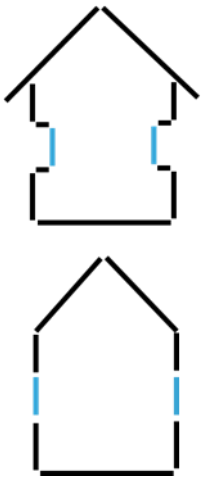
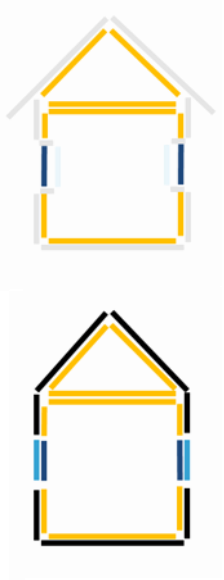
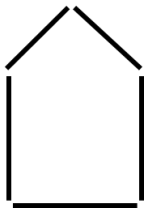
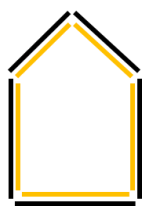


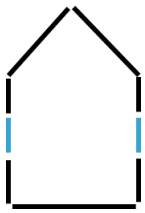
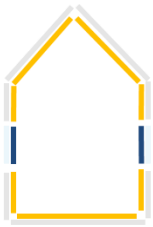
Figure 21. Example of the colour coding used in the images of Table 2.

Table 2. Examples on how to model thermal zones.

Example 1: LoD1 building. No thematic surfaces, no openings	
Case 1.1: **Single LoD1 thermal zone**, no thermal openings, no thermal boundaries • One thermal zone must be created • The thermal zone can optionally be modelled as a LoD1 solid geometry. If applicable, XLinks can be used to reference the homologous polygons of the LoD1 solid • No thermal boundaries/openings are allowed • A *LayeredConstruction* object is associated to the thermal zone: You use a single set of *averaged* building physics values (e.g. U and g values) for the whole thermal hull	
Case 1.2: **Multiple LoD1 thermal zones**, no thermal openings, no thermal boundaries • Basically, same rules as before, Case 1.1 • All thermal zones must be created • The thermal zones can optionally be modelled as LoD1 solid geometries. XLinks can be used to reference the homologous polygons of the LoD1 solid • No thermal boundaries/openings are allowed • A *LayeredConstruction* object is associated to each thermal zone: For each one, you use a single set of *averaged* building physics values (e.g. U and g values) for the thermal hull	
Example 2: LoD2 building modelled via thematic surfaces. With ADE attributes for thematic surfaces, information about azimuth, inclination, sky view factor, etc. can be stored.	

Case 2.1: **Single LoD2 thermal zone**, no thermal openings, some LoD2 thermal boundaries may have the same geometry of the corresponding building LoD2 thematic surfaces (and others don't) • One thermal zone must be created • The thermal zone can optionally be modelled as a LoD2 solid geometry. • If thermal boundaries are used, they all must be created • If the geometry of the thermal boundary and the "corresponding" thematic surface coincide, an XLinks to the LoD2 geometry of the thematic surface can be used • Optionally (and if applicable) a thermal boundary can be related to the "corresponding" building thematic surface through *RelationToCityObject* • *LayeredConstruction* objects (i.e. for U values) are associated to the thermal boundaries	
Example 3: LoD2 building modelled via thematic surfaces. With ADE attributes for thematic surfaces, information about azimuth, inclination, sky view factor, etc. can be stored.	 
Case 3.1: **Multiple LoD2 thermal zones**, no thermal openings, some LoD2 thermal boundaries may have the same geometry of the corresponding building LoD2 thematic surfaces (and others don't) • Basically, same rules as for Case 2.1 • The thermal zones can optionally be modelled as LoD2 solid geometries. If applicable, XLinks can be used to reference the homologous polygons of the LoD2 solid • If thermal boundaries are used, they all must be created • For each thermal zone, all its thermal boundaries must be created • If the geometry of the thermal boundary and the "corresponding" thematic surface coincide, an XLink to the LoD2 geometry of the thematic surface can be used • Optionally (and if applicable) a thermal boundary can be related to the "corresponding" building thematic surface through *RelationToCityObject* • *LayeredConstruction* objects (i.e. for U values) are associated to the thermal boundaries	 

Example 4: LoD3 building modelled via thematic surfaces and with openings With ADE attributes for thematic surfaces and openings, information about azimuth, inclination, sky view factor, etc. can be stored.	
Case 4.1: **Multiple LoD3 thermal zones** with thermal openings, some LoD3 thermal boundaries have the same geometry of the corresponding building LoD3 thematic surfaces/opening (and others don't) • Basically, same rules as for Case 2.1 • All thermal zones must be created • The thermal zones can optionally be modelled as a LoD3 solid geometry. If applicable, XLinks can be used to reference the homologous polygons of the LoD3 solid • If thermal boundaries and thermal openings are used, they all must be created • If the geometry of the thermal boundary and the "corresponding" building thematic surface coincide, an XLink to the LoD3 surface can be used • If the geometry of the thermal opening and the "corresponding" building opening coincide, an XLink to the LoD3 surface can be used • Optionally (and if applicable) a thermal boundary/opening can be related to the "corresponding" building thematic surface/opening through *RelationToCityObject* • *LayeredConstruction* objects (i.e. for U values) are associated to the thermal boundaries/openings	
Example 5: LoD2 building modelled via thematic surfaces. Roof edge extruded downwards to create the LoD2 envelope. No overhangs. With ADE attributes for thematic surfaces, information about azimuth, inclination, sky view factor, etc. can be stored. Real world examples: 3DBAG (NL), Vienna 3D city model (AUT), etc.	
Special case 5.1: Single LoD2 thermal zone, no thermal openings. <u>All</u> LoD2 thermal boundaries coincide <u>exactly</u> with the building LoD2 thematic surfaces • One thermal zone must be created. Set property ***coincidesWithLoD2Hull*** of the thermal zone to **TRUE** • The thermal zone can optionally be modelled as a LoD2 solid geometry. If applicable, XLinks can be used to reference the homologous polygons of the LoD2 solid • None the LoD2 thermal boundaries must be modelled. The corresponding building LoD2 thematic surfaces must be used instead • *LayeredConstruction* objects (i.e. for U and g values) are associated to the building thematic surfaces/openings.	

Example 6: LoD3 building modelled via thematic surfaces and with openings. Roof edge extruded downwards to create the LoD3 envelope. Openings are "simply" cut into the hull surfaces. No overhangs With ADE attributes for thematic surfaces and openings, information about azimuth, inclination, sky view factor, etc. can be stored.	
Special case 6.1: Single LoD3 thermal zone, with thermal boundaries and thermal openings. All LoD3 thermal boundaries/opening coincide exactly with the building LoD3 thematic surfaces/openings • Basically, same rules as for Case 5.1 • One thermal zone must be created. Set property ***coincidesWithLoD3Hull*** of the thermal zone to **TRUE** • The thermal zone can optionally be modelled as a LoD3 solid geometry. If applicable, XLinks can be used to reference the homologous polygons of the LoD3 solid • Neither the LoD3 thermal boundaries nor the LoD3 thermal openings must be modelled. The corresponding building LoD3 thematic surfaces and openings must be used instead • *LayeredConstructions* (i.e. U and g values) are associated to the building thematic surfaces/openings	

4.3.2 Using the new ADE classes for thermal boundaries

The Energy ADE 3.0 adds four new classes derived from *_BoundarySurface*: *PartyWallSurface*, *AtticFloorSurface*, *IntermediateFloorSurface*, and *BasementCeilingSurface*. Figure 22 and Figure 23 provide simple, schematic examples of when to use such new classes to model adjacent buildings and thermal zones, respectively. In these examples, class *PartyWallSurface* is represented in red and is intended to be used when modelling:

- Party walls between two adjacent buildings (e.g. between buildings A and B, and B and C)
- Party walls between two adjacent thermal zones (e.g. in building D)

Class *IntermediateFloorSurface* is represented in azure (see buildings D and E), and is intended to be used to model the horizontal surface subdividing two thermal zones, unless they correspond with the following two special cases.
Class *AtticFloorSurface* is used to model the horizontal surface of a thermal zone which is located under an “empty” space which is not modelled as a thermal zone. An example of such empty space could be an attic under the roof (see building E).
Class *BasementCeilingSurface* is used to model the horizontal surface of a thermal zone which is located above an “empty” space which is not modelled as a thermal zone. An example of such empty space could be an underground garage (see building E).

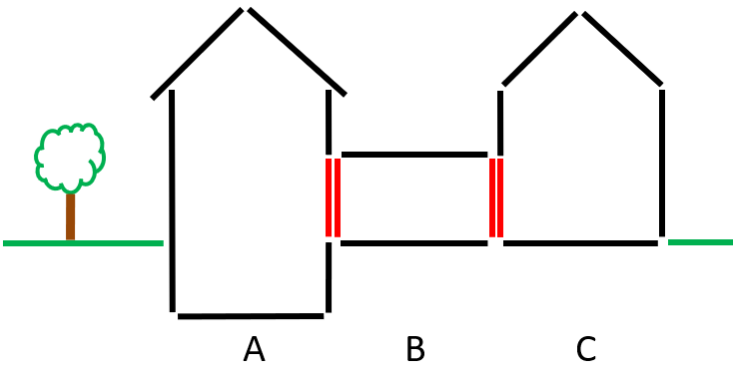


Figure 22. Schematic examples of surfaces modelled using class *PartyWallSurface* (in red).

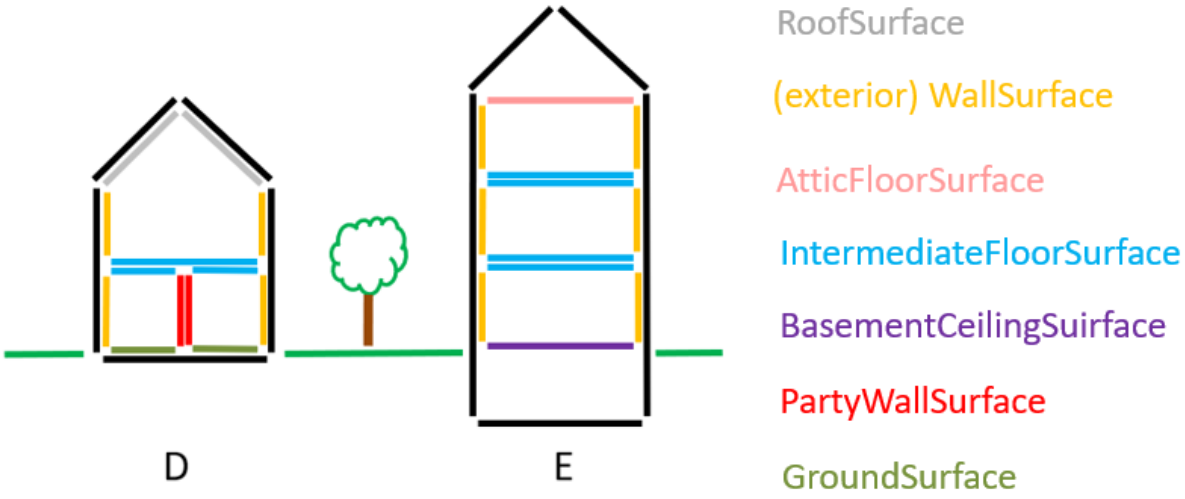


Figure 23. Schematic examples of thermal boundary surfaces modelled using different classes.

4.4 Occupancy module

4.4.1 Rule 5: Modelling and writing *Address* objects

Address objects must be always modelled as belonging to the respective parent *Building* (or *BuildingPart*) objects, as in standard CityGML. Additionally, in accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Occupancy module, the association between *BuildingUnit* and *Address* is of type “byReference”, therefore it must be written using an XLink. An example is provided in Figure 17 (line 18).

4.4.2 Modelling geometries of *UsageZone* and *BuildingUnit* objects

Rule 2 applies also to modelling the geometries of *UsageZone* and *BuildingUnit* objects. All (optional) representations by means of solid geometries (for LoD1, LoD2 and LoD3), inherited from *AbstractBuildingPartition*, can be either explicit, or references to geometries used elsewhere, but within the same building *and* the same LoD. For example, if a LoD2 usage zone corresponds geometrically to the LoD2 thermal zone that is part of, then XLinks can be used between the LoD2 solid of the usage zone and the LoD2 solid of the thermal zone.

4.5 Devices module

4.5.1 Rule 6: Modelling of shading devices

When modelling with CityGML and the Energy ADE 3.0, real-world shading devices must be grouped into two main categories: *Movable* and *immovable* shading devices.

- Immovable shading devices belonging for example to buildings must be modelled using CityGML class *BuildingInstallation*. Similar concept holds for the Bridge module, using class *BridgeInstallation*.
- Movable shading devices must be modelled as Energy ADE 3.0 *MovableShadingDevice* from the Devices module. The reason is that *MovableShadingDevices* can be associated with a *DeviceOperation* object which allows to model their time-dependent characteristics.

4.5.2 Modelling of *SolarCollector* objects

The Energy ADE 3.0 allows to optionally model solar collectors in LoD2 and LoD3, using multisurface geometries. Simplified visual examples are given in Figure 24. The different LoD representations can be used as follows:

- Clusters of solar collectors can be modelled as one single *SolarCollector* object (possibly specifying the *numberOfDevices* attribute) and using a single LoD2 geometry. This is the case, for example, when geometries of installed solar collectors are extracted from aerial imagery and may not be accurate enough to represent each single device. If the specific type of the solar collector is not known, then class *GenericSolarCollector* can be used
- If the geometry of each single solar collector is available and is of sufficient accuracy, then each collector can be modelled as a single object connected to its own LoD3 geometry. In this way, if applicable, different types of solar collectors, even on the same roof, can be represented.

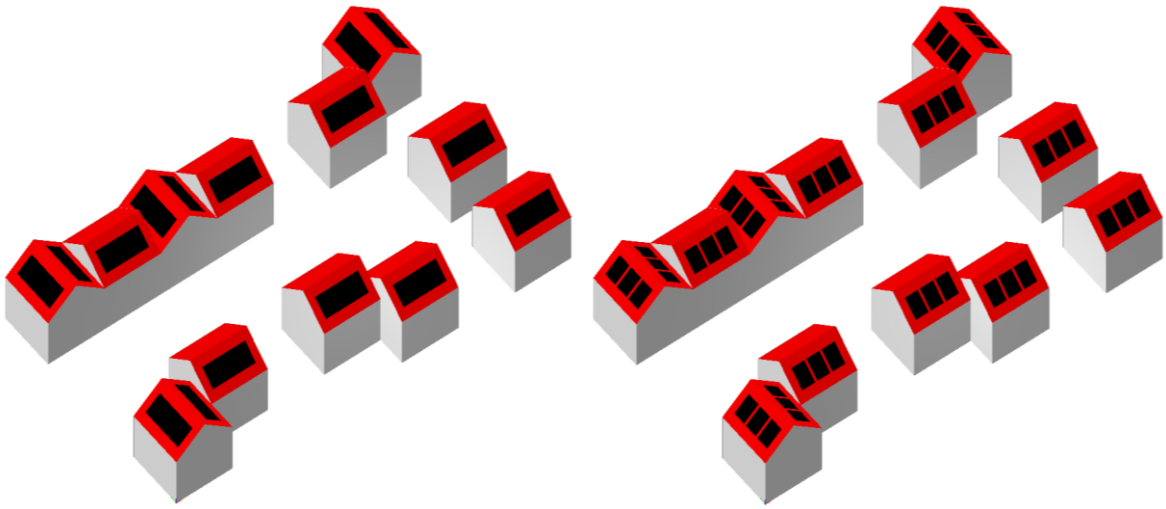


Figure 24. The black geometries on the roofs represent an example of solar collectors modelled in LoD2 [left] and in LoD3 [right].

4.6 Urban function areas module

4.6.1 Modelling *UrbanFunctionArea* objects

Save for two specific attributes, *UrbanFunctionArea* objects are indeed very similar to *CityObjectGroup* objects and follow therefore the same modelling rules. However, given the peculiar spatial connotation of *UrbanFunctionArea* objects, it is recommended that they possess a geometry too.
Additionally, the geometries of the *UrbanFunctionArea* objects should not overlap. Although gaps between nearby polygons are possible, the set of all *UrbanFunctionArea* geometries should be ideally seen as a planar partition of space, i.e. without gaps and without overlaps.

4.7 Weather station module

4.7.1 Writing *WeatherStation* elements

A *WeatherStation* element shall be written as a top-level element. If a *WeatherStation* object is installed for example on a device or a building, this relation should be modelled using the *CityObjectRelation*. An example is provided in Figure 25.

```
<core:cityObjectMember>
    <nrg3:WeatherStation gml:id="id_weather_station_1">
        <gml:description>This is Weather Station 1</gml:description>
        <gml:name>Weather Station 1</gml:name>
        ...
        <nrg3:relatedTo>
            <nrg3:CityObjectRelation>
                <nrg3:relationType codeSpace="relation_codespace">installedOn</nrg3:relationType>
                <nrg3:relatedTo xlink:href="#id_city_furniture_1"/>
            </nrg3:CityObjectRelation>
        </nrg3:relatedTo>
        ...
    </nrg3:WeatherStation>
</core:cityObjectMember>

<core:cityObjectMember>
    <frn:CityFurniture gml:id="id_city_furniture_1">
        <gml:description>Pole upon which Weather Station 1 is installed</gml:description>
        <gml:name>Weather station pole 1</gml:name>
        ...
    </frn:CityFurniture>
</core:cityObjectMember>
```

Figure 25. Excerpt of XML showing an example of how a relation can be modelled between a weather station and the pole/support (here: as *CityFurniture* object) upon which it is installed.

4.8 Layered construction module

4.8.1 Rule 7: Writing *MaterialLibrary* elements

A *MaterialLibrary* element (only one for each document) must appear as top-level element in the CityGML file. In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Layered construction module, all elements corresponding to classes derived from *AbstractMaterial*, i.e. *Gas* and *SolidMaterial*, must be written inline as children of the

MaterialLibrary element. An example is provided in

```
<core:cityObjectMember>
    <nrg3:MaterialLibrary gml:id="id_material_library_1">
        <gml:description>Example of Material Library</gml:description>
        <gml:name>Material Library 1</gml:name>
        ...
        <nrg3:libraryMember>
            <nrg3:Gas gml:id="id_gas_1">
                <gml:name>Gas 1</gml:name>
                ...
                <nrg3:isVentilated>false</nrg3:isVentilated>
                <nrg3:rValue uom="J/K/mol">8.314</nrg3:rValue>
            </nrg3:Gas>
        </nrg3:libraryMember>
        <nrg3:libraryMember>
            <nrg3:SolidMaterial gml:id="id_solid_material_1">
                <gml:name>SolidMaterial 1</gml:name>
                ...
                <nrg3:thermalConductivity uom="W/(K*m)">3.5</nrg3:thermalConductivity>
                <nrg3:density uom="kg/m^3">2500</nrg3:density>
                <nrg3:specificHeatCapacity uom="J/(kg*K)">0.9</nrg3:specificHeatCapacity>
                ...
            </nrg3:SolidMaterial>
        </nrg3:libraryMember>
        <nrg3:libraryMember>
            <nrg3:SolidMaterial gml:id="id_solid_material_2">
                <gml:name>SolidMaterial 2</gml:name>
                ...
            </nrg3:SolidMaterial>
        </nrg3:libraryMember>
    </nrg3:MaterialLibrary>
</core:cityObjectMember>
```

Figure 26.


```
<core:cityObjectMember>
    <nrg3:MaterialLibrary gml:id="id_material_library_1">
        <gml:description>Example of Material Library</gml:description>
        <gml:name>Material Library 1</gml:name>
        ...
        <nrg3:libraryMember>
            <nrg3:Gas gml:id="id_gas_1">
                <gml:name>Gas 1</gml:name>
                ...
                <nrg3:isVentilated>false</nrg3:isVentilated>
                <nrg3:rValue uom="J/K/mol">8.314</nrg3:rValue>
            </nrg3:Gas>
        </nrg3:libraryMember>
        <nrg3:libraryMember>
            <nrg3:SolidMaterial gml:id="id_solid_material_1">
                <gml:name>SolidMaterial 1</gml:name>
                ...
                <nrg3:thermalConductivity uom="W/(K*m)">3.5</nrg3:thermalConductivity>
                <nrg3:density uom="kg/m^3">2500</nrg3:density>
                <nrg3:specificHeatCapacity uom="J/(kg*K)">0.9</nrg3:specificHeatCapacity>
                ...
            </nrg3:SolidMaterial>
        </nrg3:libraryMember>
        <nrg3:libraryMember>
            <nrg3:SolidMaterial gml:id="id_solid_material_2">
                <gml:name>SolidMaterial 2</gml:name>
                ...
            </nrg3:SolidMaterial>
        </nrg3:libraryMember>
    </nrg3:MaterialLibrary>
</core:cityObjectMember>
```


Figure 26. Excerpt of XML showing an example of a MaterialLibrary element containing one *Gas* object and two *SolidMaterial* elements.

4.8.2 Rule 8: Writing *LayeredConstructionLibrary* elements

LayeredConstructionLibrary elements must appear as top-level elements in the CityGML file. In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Layered construction module, all elements corresponding to classes *LayeredConstruction* or *ReverseLayeredConstruction* must be written inline as children of a *LayeredConstructionLibrary* element. An example is provided in Figure 27.

```
4  <core:cityObjectMember>
5      <nrg3:LayeredConstructionLibrary gml:id="id_layered_construction_library_1">
6          <gml:description>Example of a Layered Construction Library</gml:description>
7          <gml:name>Layered Construction Library 1</gml:name>
8          ...
9          <nrg3:libraryMember>
10             <nrg3:LayeredConstruction gml:id="id_layered_construction_glazing_5">
11                 <gml:description>Example of a LayeredConstruction (without layers)</gml:description>
12                 ...
13             </nrg3:LayeredConstruction>
14         </nrg3:libraryMember>
15
16         <nrg3:libraryMember>
17             <nrg3:LayeredConstruction gml:id="id_layered_construction_ground_1">
18                 <gml:description>Example of a LayeredConstruction (with 2 layers)</gml:description>
19                 ...
20                 <nrg3:layer>
21                     <nrg3:Layer gml:id="id_layer_1">
22                         ...
23                         <nrg3:thickness uom="mm">40</nrg3:thickness>
24                         <nrg3:material xlink:href="#id_solid_material_1"/>
25                     </nrg3:Layer>
26                 </nrg3:layer>
27                 <nrg3:layer>
28                     <nrg3:Layer gml:id="id_layer_2">
29                         ...
30                         <nrg3:thickness uom="mm">40</nrg3:thickness>
31                         <nrg3:material xlink:href="#id_solid_material_2"/>
32                     </nrg3:Layer>
33                 </nrg3:layer>
34             </nrg3:LayeredConstruction>
35         </nrg3:libraryMember>
36     </nrg3:LayeredConstructionLibrary>
37 </core:cityObjectMember>
```

Figure 27. Excerpt of XML showing an example of a *LayeredConstructionLibrary* element containing two *LayeredConstruction* elements: the first one without layers, the second one with two layers.

4.8.3 Modelling *LayeredConstruction* objects

A *LayeredConstruction* object can be modelled as a simple object containing, for example, the unique average U-value of the wall surface it refers to. The U-value represents the global U-value of the series of layers it is made of, even if no layer information is provided. Alternatively, if more information is available, its component layers can be specified too. The order of the layers follows the direction of the normal vector of the surface they refer to. An example is provided in in Figure 27.

4.8.4 Writing *Layer* elements

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Layered construction module, each *Layer* element must be written inline as a child of the containing *LayeredConstruction* element. Additionally, each *Layer* object must express the association with the corresponding material (through property *material*) by means of an XLink reference. An example is provided in Figure 27 (e.g. lines 23 and 30).

4.8.5 Modelling the association from *ReverseLayeredConstruction* to *LayeredConstruction*

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Layered construction module, the *baseLayeredConstructon* property of a *ReverseLayeredConstruction* object is used to indicate the reference *LayeredConstruction* object for which the order of the layers must be inverted. The association to *LayeredConstruction* object is of type "byReference", therefore it must be written by mean of an XLink. Please note that a *ReverseLayeredConstruction* element, just like a *LayeredConstruction* one, must be written inline as a direct child of a *LayeredConstructionLibrary* element.

```
<core:cityObjectMember>
    <nrg3:LayeredConstructionLibrary gml:id="id_layered_construction_library_2">
        <gml:description>Example of a Layered Construction Library</gml:description>
        <gml:name>Layered Construction Library 2</gml:name>
        ...
        <nrg3:libraryMember>
            <nrg3:LayeredConstruction gml:id="id_layered_construction_wall_2">
                ...
                <nrg3:layer>
                    <nrg3:Layer gml:id="id_layer_1">
                        ...
                    </nrg3:Layer>
                </nrg3:layer>
                <nrg3:layer>
                    <nrg3:Layer gml:id="id_layer_2">
                        ...
                    </nrg3:Layer>
                </nrg3:layer>
                <nrg3:layer>
                    <nrg3:Layer gml:id="id_layer_3">
                        ...
                    </nrg3:Layer>
                </nrg3:layer>
            </nrg3:LayeredConstruction>
        </nrg3:libraryMember>
        <nrg3:libraryMember>
            <nrg3:ReverseLayeredConstruction gml:id="id_reverse_layered_construction_wall_2">
                <gml:description>Example of a Reverse Layered Construction</gml:description>
                ...
                <nrg3:baseLayeredConstruction xlink:href="#id_layered_construction_wall_2"/>
            </nrg3:ReverseLayeredConstruction>
        </nrg3:libraryMember>
    </nrg3:LayeredConstructionLibrary>
</core:cityObjectMember>
```

Figure 28. Excerpt of XML showing an example of a *LayeredConstructionLibrary* element containing two *LayeredConstruction* elements: the first one, a *LayeredConstruction*, contains an ordered set of layers. The second one, a *ReverseLayeredConstruction*, references the first one by means of an XLink.

4.9 Resources module

4.9.1 Writing resource objects

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Core module, all resource elements corresponding to any subclass of *AbstractResource* must be written as inline objects of the respective containing city object element. Additionally, and if applicable, the element corresponding to any subclass of *AbstractTimeSeries* must be written inline as child of the containing resource element. An example is provided in Figure 29.

```
<core:cityObjectMember>
    <nrg3:WeatherStation gml:id="id_weather_station_1">
        <gml:name>Weather Station 1</gml:name>
        ...
        <nrg3:resource>
            <nrg3:Energy gml:id="id_res_energy_1">
                <gml:description>Electricity demand</gml:description>
                ...
                <nrg3:timeDependentAmount>
                    <nrg3:RegularTimeSeries gml:id="id_time_series_1">
                    ...
                    </nrg3:RegularTimeSeries>
                </nrg3:timeDependentAmount>
                ...
            </nrg3:Energy>
        </nrg3:resource>
        ...
    </nrg3:WeatherStation>
</core:cityObjectMember>
```

Figure 29. Excerpt of XML showing an example of a *WeatherStation* element containing an *Energy* resource.

4.10 Schedules module

4.10.1 Rule 9: Writing *ScheduleLibrary* elements

A *ScheduleLibrary* element (ideally only one for each city model) must appear as top-level element in the CityGML file. In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Schedules module, all elements corresponding to subclasses of *AbstractSchedule*, i.e. *AtomicSchedule*, *DualValueSchedule*, and *CompositeSchedule*, must be written inline as children of the *ScheduleLibrary* element. An example is provided in Figure 30.

```
<core:cityObjectMember>
    <nrg3:ScheduleLibrary gml:id="id_schedule_library_1">
        <gml:description>This is Schedule Library 1</gml:description>
        <gml:name>Schedule Library 1</gml:name>
        ...
        <nrg3:libraryMember>
            <nrg3:AtomicSchedule gml:id="id_atomic_schedule_1">
                ...
            </nrg3:AtomicSchedule>
        </nrg3:libraryMember>

        <nrg3:libraryMember>
            <nrg3:CompositeSchedule gml:id="id_composite_schedule_3">
            ...
            </nrg3:CompositeSchedule>
         </nrg3:libraryMember>

        <nrg3:libraryMember>
            <nrg3:DualValueSchedule gml:id="id_dual_value_schedule_2">
                ...
            </nrg3:DualValueSchedule>
        </nrg3:libraryMember>

    </nrg3:ScheduleLibrary>
</core:cityObjectMember>
```

Figure 30. Excerpt of XML showing an example of a *ScheduleLibrary* element containing an *AtomicSchedule* element, a *DualValueSchedule* element and a *CompositeSchedule* element.

4.10.2 Modelling of *ScheduleComponent* objects

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the Schedules module, all *ScheduleComponent* elements must be written inline as children of the containing *CompositeSchedule* element. Additionally, each *ScheduleComponent* element must express the association with the corresponding target schedule (through property *scheduleComponentMember*) by means of an XLink. An example is provided in Figure 31.

```
<nrg3:ScheduleLibrary gml:id="id_schedule_library_2">
        ...
    <nrg3:libraryMember>
        <!-- Atomic Schedule of typical week day -->
        <nrg3:AtomicSchedule gml:id="id_atomic_schedule_10">
            <gml:description>Typical weekday daily schedule</gml:description>
            ...
            <nrg3:type codeSpace="schedule_type_codeSpace">weekDay</nrg3:type>
            <nrg3:startTime>00:00:00</nrg3:startTime>
            <nrg3:temporalExtent unit="day">1</nrg3:temporalExtent>
            <nrg3:timeSeries>
                <nrg3:TypicalValuesTimeSeries gml:id="id_time_series_21">
                    ...
                </nrg3:TypicalValuesTimeSeries>
            </nrg3:timeSeries>
        </nrg3:AtomicSchedule>
    </nrg3:libraryMember>
    <nrg3:libraryMember>
        <!-- Atomic Schedule of typical weekend day -->
        <nrg3:AtomicSchedule gml:id="id_atomic_schedule_11">
            <gml:description>Typical weekend daily schedule</gml:description>
            ...
            <nrg3:type codeSpace="schedule_type_codeSpace">weekendDay</nrg3:type>
            <nrg3:startTime>00:00:00</nrg3:startTime>
            <nrg3:temporalExtent unit="day">1</nrg3:temporalExtent>
            <nrg3:timeSeries>
                <nrg3:TypicalValuesTimeSeries gml:id="id_time_series_22">
                    ...
                </nrg3:TypicalValuesTimeSeries>
            </nrg3:timeSeries>
        </nrg3:AtomicSchedule>
    </nrg3:libraryMember>
    <nrg3:libraryMember>
        <!-- Composite Schedule of typical week -->
        <nrg3:CompositeSchedule gml:id="id_composite_schedule_12">
            <gml:description>Typical week: 5 week days + 2 weekend days</gml:description>
            ...
            <nrg3:type codeSpace="schedule_type_codeSpace">week</nrg3:type>
            <nrg3:startTime>00:00:00</nrg3:startTime>
            <nrg3:temporalExtent unit="day">7</nrg3:temporalExtent>
            <nrg3:scheduleComponent>
                <nrg3:ScheduleComponent gml:id="id_schedule_component_1">
                    ...
                    <nrg3:repetitions>5</nrg3:repetitions>
                    <nrg3:additionalGap unit="day">0</nrg3:additionalGap>
                    <nrg3:scheduleComponentMember xlink:href="#id_atomic_schedule_10"/>
                </nrg3:ScheduleComponent>
            </nrg3:scheduleComponent>
            <nrg3:scheduleComponent>
                <nrg3:ScheduleComponent gml:id="id_schedule_component_2">
                    ...
                    <nrg3:repetitions>2</nrg3:repetitions>
                    <nrg3:additionalGap unit="day">0</nrg3:additionalGap>
                    <nrg3:scheduleComponentMember xlink:href="#id_atomic_schedule_11"/>
                </nrg3:ScheduleComponent>
            </nrg3:scheduleComponent>
        </nrg3:CompositeSchedule>
    </nrg3:libraryMember>
</nrg3:ScheduleLibrary>
```

Figure 31. Excerpt of XML showing an example of a *CompositeSchedule* element composed of two *ScheduleComponent* elements, each one referring to an *AtomicSchedule* element by means of XLinks.

4.11 Time series module

4.11.1 Modelling of time series objects

In accordance with **Rule 1** and as suggested by the annotations in the UML diagrams of the different modules, all elements corresponding to any subclass of *AbstractTimeSeries* must be written inline as children of the respective containing elements. Examples are provided in Figure 29 (line 14) and Figure 31 (lines 17 and 32).

5 Derivation of the XSD file

XML Schema Definition (XSD) files encode, in a machine-readable way, a data model and its constraints, therefore defining how data can be written and automatically validated. For CityGML, the XSD files are already available and published on the respective CityGML Wiki page[16].

Once the UML-based modelling of the Energy ADE 3.0 has been completed, an XSD file has been derived from the UML-based data model in order to create and validate instance documents containing CityGML and Energy ADE 3.0 data. The XSD file has been first automatically derived using the software tool ShapeChange v. 3.10[17]. ShapeChange requires a custom configuration file. For this purpose, an already existing configuration file from the Utility Network ADE[18] has been adapted to match the requirements of ShapeChange 3.10 and of the Energy ADE 3.0. Successively, the resulting XSD file has been carefully manually checked to ensure the correctness of the classes and properties.
Additionally, sets of tagged values have been manually added to certain elements of the XSD file. As a matter of fact, during the generation of the XSD of a CityGML ADE relevant information about the ADE data model is often missing. Examples of such missing information refer for example to the type of associations or the multiplicity of attributes contained by ADE classes that extend existing CityGML classes by means of the ADE-hook mechanism.
This additional information plays a major role in the successive step when the DDL scripts for the database schema are generated, allowing to automatically generate tables, constraints, indices, and functions. In particular, depending on the type of association, different delete functions will be generated. More details can be found in the online 3DCityDB documentation[19].
In order to overcome such shortcomings, the type of associations (e.g. simple association, aggregation, or composition) and other details have been manually added via sets of tagged values to the XSD file of the Energy ADE 3.0. Figure 32 shows a screenshot of the first lines of the resulting Energy ADE 3.0 XSD file. The derived XSD schema file has been then used both in Safe Software's FME Form and in the KIT ModelViewer for testing purposes, to generate and visualize test datasets (see section 6) and to generate the database DDL scripts (see section 7).

[16] https://www.citygmlwiki.org/index.php/CityGML_XML_Schema
[17] https://github.com/ShapeChange/ShapeChange/releases/tag/3.1.0
[18] https://github.com/TatjanaKutzner/CityGML-UtilityNetwork-ADE/releases/tag/v0.9.3
[19] https://3dcitydb-docs.readthedocs.io/en/latest/plugins/ade-manager/tagged-values.html

```
<schema
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml"
xmlns:app="http://www.opengis.net/citygml/appearance/2.0"
xmlns:core="http://www.opengis.net/citygml/2.0"
xmlns:brid="http://www.opengis.net/citygml/bridge/2.0"
xmlns:bldg="http://www.opengis.net/citygml/building/2.0"
xmlns:frn="http://www.opengis.net/citygml/cityfurniture/2.0"
xmlns:grp="http://www.opengis.net/citygml/cityobjectgroup/2.0"
xmlns:gen="http://www.opengis.net/citygml/generics/2.0"
xmlns:luse="http://www.opengis.net/citygml/landuse/2.0"
xmlns:dem="http://www.opengis.net/citygml/relief/2.0"
xmlns:tran="http://www.opengis.net/citygml/transportation/2.0"
xmlns:tun="http://www.opengis.net/citygml/tunnel/2.0"
xmlns:veg="http://www.opengis.net/citygml/vegetation/2.0"
xmlns:wtr="http://www.opengis.net/citygml/waterbody/2.0"
xmlns:nrg3="http://www.citygml.org/ade/energy/3.0"
elementFormDefault="qualified"
targetNamespace="http://www.citygml.org/ade/energy/3.0"
version="3.0"
>
  <annotation>
    <documentation>Energy ADE 3.0 beta 7 for CityGML 2.0</documentation>
  </annotation>

  <!-- All CityGML modules added manually, in case they are not written automatically by ShapeChange -->

  <import namespace="http://www.opengis.net/gml" schemaLocation="http://schemas.opengis.net/gml/3.1.1/bas
  <import namespace="http://www.opengis.net/citygml/2.0" schemaLocation="http://schemas.opengis.net/cityg
  <import namespace="http://www.opengis.net/citygml/appearance/2.0" schemaLocation="http://schemas.opengi
  <import namespace="http://www.opengis.net/citygml/bridge/2.0" schemaLocation="http://schemas.opengis.ne
  <import namespace="http://www.opengis.net/citygml/building/2.0" schemaLocation="http://schemas.opengis.
  <import namespace="http://www.opengis.net/citygml/cityfurniture/2.0" schemaLocation="http://schemas.ope
  <import namespace="http://www.opengis.net/citygml/cityobjectgroup/2.0" schemaLocation="http://schemas.c
  <import namespace="http://www.opengis.net/citygml/generics/2.0" schemaLocation="http://schemas.opengis.
  <import namespace="http://www.opengis.net/citygml/landuse/2.0" schemaLocation="http://schemas.opengis.n
  <import namespace="http://www.opengis.net/citygml/relief/2.0" schemaLocation="http://schemas.opengis.ne
  <import namespace="http://www.opengis.net/citygml/transportation/2.0" schemaLocation="http://schemas.op
  <import namespace="http://www.opengis.net/citygml/tunnel/2.0" schemaLocation="http://schemas.opengis.ne
  <import namespace="http://www.opengis.net/citygml/vegetation/2.0" schemaLocation="http://schemas.opengi
  <import namespace="http://www.opengis.net/citygml/waterbody/2.0" schemaLocation="http://schemas.opengis

  <!-- ********************** -->
  <!-- Energy ADE Enumerations -->
  <!-- ********************** -->

  <simpleType name="InterpolationTypeValueType">
    <restriction base="string">
      <enumeration value="averageInPrecedingInterval"></enumeration>
      <enumeration value="averageInSucceedingInterval"></enumeration>
      <enumeration value="constantInPrecedingInterval"></enumeration>
      <enumeration value="constantInSucceedingInterval"></enumeration>
      <enumeration value="continuous"></enumeration>
      <enumeration value="discontinuous"></enumeration>
      <enumeration value="instantaneousTotal"></enumeration>
      <enumeration value="maximumInPrecedingInterval"></enumeration>
      <enumeration value="maximumInSucceedingInterval"></enumeration>
      <enumeration value="minimumInPrecedingInterval"></enumeration>
      <enumeration value="minimumInSucceedingInterval"></enumeration>
      <enumeration value="precedingTotal"></enumeration>
      <enumeration value="succeedingTotal"></enumeration>
    </restriction>
  </simpleType>

  <simpleType name="NetworkConnectionStatusValueType">
    <restriction base="string">
      <enumeration value="connected"></enumeration>
      <enumeration value="disconnectedButConnectable"></enumeration>
      <enumeration value="disconnectedAndNotConnectable"></enumeration>
      <enumeration value="unknown"></enumeration>
    </restriction>
  </simpleType>
```

Figure 32. Screenshot of an excerpt of the XSD file of the Energy ADE 3.0.

6 Test datasets

Safe Software's FME Form 2024, together with the XSD file of the Energy ADE 3.0, has been used to generate some test datasets. Tests have been iteratively carried out to verify the correctness of the generated XSD file to create the desired FME Readers and Writers for CityGML. During the development process of the Energy ADE 3.0, this iterative process has helped in the identification of errors and to improve the conceptual model.

Several test datasets have been created. They represent a starting point for testing the following implementation steps (e.g. the Java-based ones, and the database schema extending the 3D City Database), but also for getting an insight in how Energy ADE 3.0 is written according to the XML encoding.

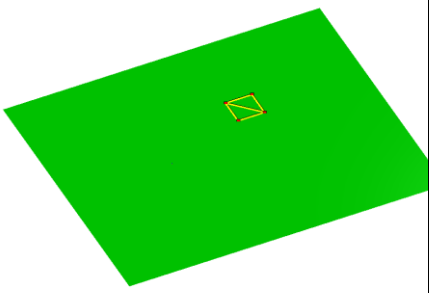
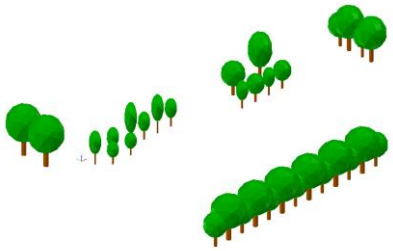
The base dataset, which has been progressively enriched with different Energy ADE 3.0 data, is a set of fictitious buildings, trees and a DTM, belonging to the dataset "Alderaan" [20], which is already available as open (test) data[21].
All "Alderaan" datasets use the Amersfoort / RD New coordinate system (EPSG: 28992[22]) for the horizontal datum, and the Normaal Amsterdams Peil (NAP) height (EPSG:5109[23]) for the vertical datum. Since the datasets are actually located near the origin of the axes, they will be visualised in France, near the municipality of Joigny.

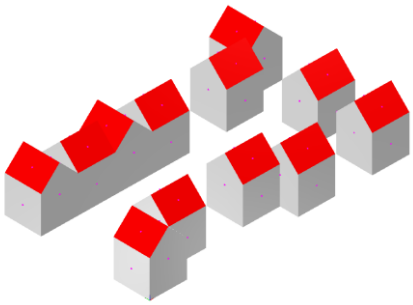
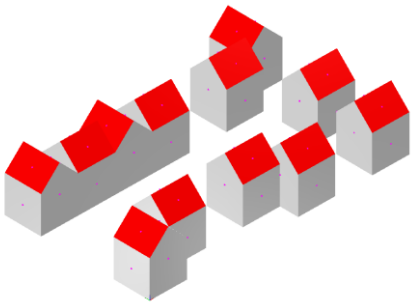
BEWARE: It is important to remark that the following datasets have been created only for testing purposes. Therefore, some parameters may have no real physical meaning, or could simply be unrealistic. Nevertheless, the purpose is to provide examples of objects of nearly every class and every attribute of the Energy ADE 3.0.

The available datasets are listed and briefly described in Table 3. The screenshots are taken using the KIT ModelViewer[24]. To load the datasets in the KIT ModelViewer, the XSD file of the Energy ADE 3.0 is required and must be copied into the \GMLSchemata\CityGML_2_0\CityGML folder of the installation folder. In this way, any CityGML file containing Energy ADE 3.0 data will be opened and the Energy ADE 3.0 data (objects with geometries and attributes) will be loaded and visualised. Please note that **KIT Modelviewer version 7.4.0** (or higher) is recommended.
All datasets are available for download from the GitHub repository[25] of the 3D Geoinformation group at TUD Delft.

[20] https://3d.bk.tudelft.nl/gagugiaro/tutorials/pdf/Alderaan_dataset.pdf
[21] https://3d.bk.tudelft.nl/gagugiaro/tutorials/pdf/CityGML_Alderaan.zip
[22] https://epsg.io/28992
[23] https://epsg.io/5109-datum
[24] https://www.iai.kit.edu/english/4561.php
[25] https://github.com/tudelft3d/Energy_ADE

Table 3. Available test datasets for the CityGML Energy ADE 3.0. For each dataset, the name of the corresponding file, the list of the CityGML and Energy ADE 3.0 classes for which objects have been created, as well as a brief description are provided.

Alderaan_DTM.gml CityGML: • ReliefFeature • TINRelief The file contains a TIN-based DTM that spans the extents of the Alderaan dataset. It is a very simple, flat DTM, consisting of 120 LoD1 *TinRelief* objects contained within 1 LoD1 *ReliefFeature* object. This file contains no Energy ADE 3.0 objects.	
Alderaan_Energy_ADE_Trees.gml CityGML: • CityObjectGroup • SolitaryVegetationObject • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject • Water The file consists of a *CityObjectGroup* object which contains 33 trees modelled in LoD1, LoD2 and LoD3. Additionally, they are represented via reference points. The *CityObjectGroup* has a *Water* object. The same applies to some trees. The *Water* objects symbolise the amount of water needed for irrigation. This datasets uses classes from the Energy ADE 3.0 Core and Resources modules.	

Alderaan_Energy_ADE_Core.gml CityGML: • Building, BuildingPart • (Building) GroundSurface, WallSurface, RoofSurface • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject, CityObjectRelation • ADE _AbstractBuilding • UtilityNetworkConnection • EnergyPerformanceCertificate • RefurbishmentMeasure The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled as LoD0 footprints, LoD1 and LoD2 solids, as well as through thematic surfaces. Additionally, they are represented via reference points. The buildings possess some ADE attributes, as well as some *UtilityNetworkConnection*, *EnergyPerformanceCertificate*, *RefurbishmentMeasure* objects. This datasets uses classes from the Energy ADE 3.0 Core module.	
Alderaan_Energy_ADE_Core_PartyWallSurfaces.gml CityGML: • Building, BuildingPart • (Building) GroundSurface, WallSurface, RoofSurface • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject, CityObjectRelation • ADE _AbstractBuilding • UtilityNetworkConnection • EnergyPerformanceCertificate • RefurbishmentMeasure • PartyWallSurface This file is the same as the previous one, with the only addition that party walls between adjacent buildings are modelled using *PartyWallSurface* objects in LoD2. This datasets uses classes from the Energy ADE 3.0 Core and Building physics module.	

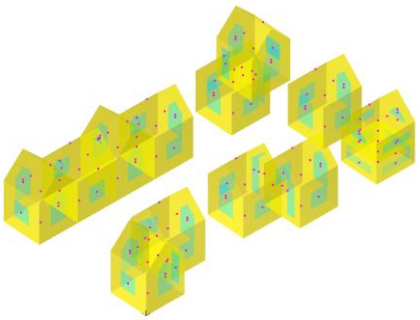
Alderaan_Energy_ADE_Core_Building_physics.gml

CityGML:
- Building, BuildingPart
- (Building) GroundSurface, WallSurface, RoofSurface
- (Building) Window
- Appearance, X3DMaterial

Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- ADE _BoundarySurface, ADE _Opening
- ThermalZone
- BasementCeilingSurface, PartyWallSurface, IntermediateFloorSurface

The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled as LoD0 footprints, LoD1 and LoD2 solids, as well as through thematic surfaces. Party walls between adjacent buildings are modelled using *PartyWallSurface* objects. Additionally, there are *ThermalZone* objects, which are modelled via solid geometries (LoD1, LoD2 and LoD3) and via thermal boundaries (LoD2 and LoD3) and thermal openings (LoD3).

This datasets uses classes from the Energy ADE 3.0 Core and Building physics module.



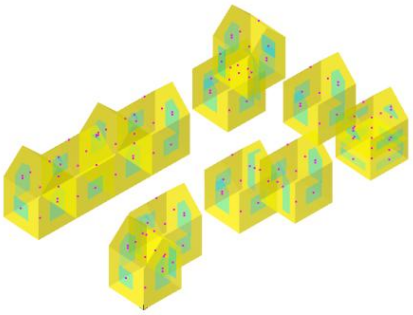
Alderaan_Energy_ADE_Core_Building_physics_LayeredConstruction.gml

CityGML:
- Building, BuildingPart
- (Building) GroundSurface, WallSurface, RoofSurface
- (Building) Window
- Appearance, X3DMaterial

Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- ADE _BoundarySurface, ADE _Opening
- ThermalZone
- BasementCeilingSurface, PartyWallSurface, IntermediateFloorSurface
- LayeredConstructionLibrary, LayeredConstruction, ReverseLayeredConstruction, Layer
- MaterialLibrary, Gas, SolidMaterial

This file is the same as the previous one, with the only addition that all thematic surfaces, thermal boundaries and openings have an XLink to a (*Reverse*)*LayeredConstruction* object.

This datasets uses classes from the Energy ADE 3.0 Core, Building physics and Layered construction modules.



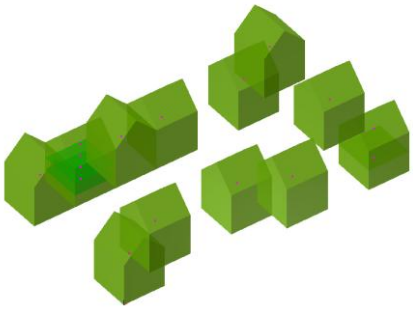
Alderaan_Energy_ADE_Core_Building_physics_UsageZone.gml

CityGML:
- Building, BuildingPart
- (Building) GroundSurface, WallSurface, RoofSurface
- (Building) Window
- Appearance, X3DMaterial

Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- ADE _BoundarySurface, ADE _Opening
- ThermalZone
- BasementCeilingSurface, PartyWallSurface, IntermediateFloorSurface
- UsageZone

The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled as LoD0 footprints, LoD1 and LoD2 solids, as well as through thematic surfaces in LoD2. Party walls between adjacent buildings are modelled using *PartyWallSurface* objects. Additionally, there are *ThermalZone* objects, which are modelled via solid geometries (LoD1, LoD2 and LoD3) and via thermal boundaries (LoD2 and LoD3) and thermal openings (LoD3). Finally, there are *UsageZone* objects, modelled as solids in LoD1, LoD2 and LoD3.

This datasets uses classes from the Energy ADE 3.0 Core, Building physics and Occupancy modules.



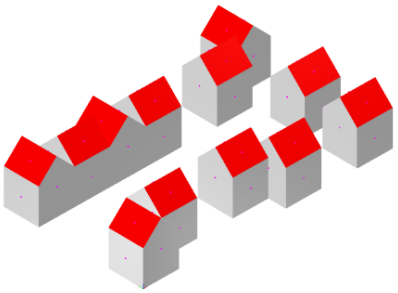
Alderaan_Energy_ADE_Core_Building_physics_ CoincidesWithLoD2Hull.gml

CityGML:
- Building, BuildingPart
- (Building) GroundSurface, WallSurface, RoofSurface
- Appearance, X3DMaterial

Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- PartyWallSurface
- ThermalZone

The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled as LoD0 footprints, LoD1 and LoD2 solids, as well as through thematic surfaces. Party walls between adjacent buildings are modelled using *PartyWallSurface* objects. Additionally, there are *ThermalZone* objects. The thermal boundaries are not defined explicitly, instead each thermal zone has the *coincidesWithLoD2Hull* attribute set to TRUE.

This datasets uses classes from the Energy ADE 3.0 Core and Building physics module.



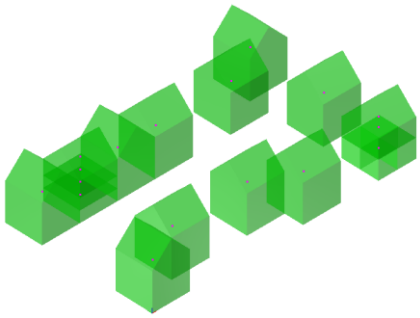
Alderaan_Energy_ADE_Core_Occupancy.gml

CityGML:
- Building, BuildingPart
- Address
- Appearance, X3DMaterial

Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- UsageZone
- BuildingUnit
- Occupants
- EnergyPerformanceCertificate
- RefurbishmentMeasure

The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled only as 3D reference point. There are *UsageZone*, *BuildingUnit*, and *Occupants* objects. *UsageZone* and *BuildingUnit* objects are modelled via solid geometries (LoD1, LoD2 and LoD3). Some of them contain objects of classes *EnergyPerformanceCertificate*, and *RefurbishmentMeasure*.

This dataset uses classes from the Energy ADE 3.0 Core and Occupancy modules.



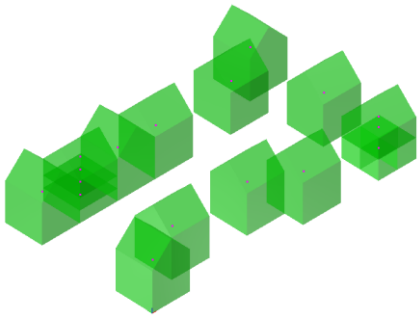
Alderaan_Energy_ADE_Core_Occupancy_Schedules.gml

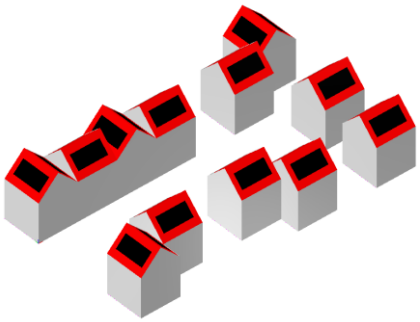
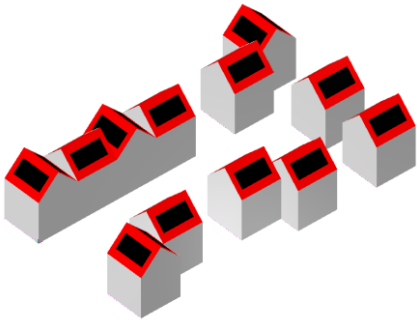
CityGML:
- Building, BuildingPart
- Address
- Appearance, X3DMaterial

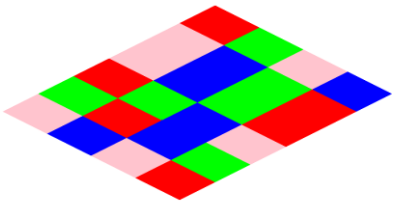
Energy ADE 3.0:
- ADE _CityObject, CityObjectRelation
- UsageZone
- BuildingUnit
- Occupants
- EnergyPerformanceCertificate
- RefurbishmentMeasure
- ScheduleLibrary, AtomicSchedule, DualValueSchedule, CompositeSchedule, ScheduleComponent
- TypicalValuesTimeSeries

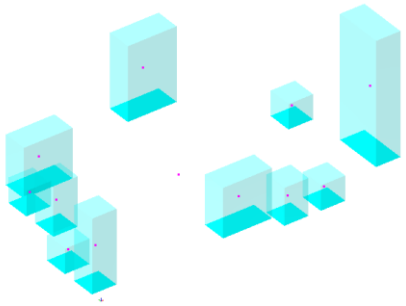
This file is the same as the previous one, with the only addition that all *UsageZone* and *Occupants* objects are associated with a schedule by means of an XLink. Different types of schedules are contained in the *ScheduleLibrary*. *AtomicSchedule* objects possess a *TypicalValuesTimeSeries*.

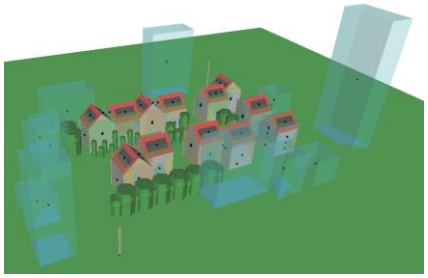
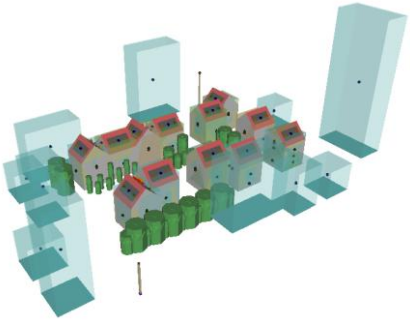
This dataset uses classes from the Energy ADE 3.0 Core, Occupancy, Schedules and Time series modules.



Alderaan_Energy_ADE_Core_Devices.gml CityGML: • Building, BuildingPart • (Building) GroundSurface, WallSurface, RoofSurface • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject, CityObjectRelation • Boiler, HeatPump, LightingDevice, GenericEletricalDevice, GenericDevice, MovableShadingDevice, ElectricalStorageDevice, ThermalStorageDevice, GenericSolarCollector, SolarThermalCollector, PhotovoltaicCollector, PhotovoltaicThermalCollector The file consists of 11 buildings. 10 are single-part buildings, 1 is a multi-part building, composed of 2 building parts. Building(Part)s are modelled through thematic surfaces in LoD2 and as 3D reference point. Each building has some *Device* objects (boilers, different types of solar collectors, electrical and thermal storage devices, etc.). All devices, except the solar collectors, have no geometries besides a reference point. The solar collectors are modelled in LoD2 and LoD3. This dataset uses classes from the Energy ADE 3.0 Core and Devices modules.	
Alderaan_Energy_ADE_Core_Devices_Schedules.gml CityGML: • Building, BuildingPart • (Building) GroundSurface, WallSurface, RoofSurface • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject, CityObjectRelation • Boiler, HeatPump, LightingDevice, GenericEletricalDevice, GenericDevice, MovableShadingDevice, ElectricalStorageDevice, ThermalStorageDevice, GenericSolarCollector, SolarThermalCollector, PhotovoltaicCollector, PhotovoltaicThermalCollector • DeviceOperation • ScheduleLibrary, AtomicSchedule, DualValueSchedule, CompositeSchedule, ScheduleComponent • TypicalValuesTimeSeries This file is the same as the previous one, with the only addition that each device has a *DeviceOperation* object which is associated to a schedule by means of an XLink. Different types of schedules are contained in the *ScheduleLibrary*. *AtomicSchedule* objects possess a *TypicalValuesTimeSeries*. This dataset uses classes from the Energy ADE 3.0 Core, Devices, Schedules and Time series modules.	

Alderaan_ADE_UrbanFunctionAreas.gml CityGML: • Appearance, X3DMaterial Energy ADE 3.0: • ADE _CityObject • UrbanFunctionArea • Energy, Water This file contains 31 *UrbanFunctionArea* objects, organised in 3 hierarchical levels. The first-level object contains 6 second-level objects, which again contain each 4 third-level objects. The image on the right represents only the 24 third-level objects. Each *UrbanFunctionArea* object is represented via a multisurface geometry and a reference point. Some *Energy* and *Water* objects are associated with the *UrbanFunctionArea* objects, in order to provide an example of resources associated at each level. This dataset uses classes from the Energy ADE 3.0 Core, Urban function areas and Resources modules.	
Alderaan_ADE_WeatherStation.gml CityGML: • CityFurniture Energy ADE 3.0: • ADE _CityObject, CityObjectRelation • WeatherData • WeatherStation • Energy • RegularTimeSeries, RegularTimeSeriesFile, TypicalValuesTimeSeries, TypicalValuesTimeSeries, SensorConnection This file contains 2 *CityFurniture* objects that represent two “poles” upon which 2 *WeatherStation* objects are installed. The *CityFurniture* objects are represented via a LoD1 geometry and a reference point. The 2 *WeatherStation* objects are represented by means of a reference point. Each weather station contains several *WeatherData* objects, which are connected to one type of time series. Finally, each weather station contains an *Energy* object, symbolising the electricity demand required by the station. One of the *Energy* objects is connected to a time series. This dataset uses classes from the Energy ADE 3.0 Core, Weather station, Resources and Time series modules.	

Alderaan_Energy_ADE_Material_LayeredConstruction_Libraries.gml Energy ADE 3.0: • LayeredConstructionLibrary, LayeredConstruction, ReverseLayeredConstruction, Layer • MaterialLibrary, SolidMaterial, Gas This file contains 2 libraries. The first one is a *MaterialLibrary* object which contains 17 *SolidMaterial* and 2 *Gas* objects. The second library contains 5 *LayeredConstruction* and 2 *ReverseLayeredConstruction* objects. Some *LayeredConstruction* objects have information about the layer structure, some don't. In case layers are modelled, they contain a reference to the corresponding material. This dataset uses classes from the Energy ADE 3.0 Layered construction module.	
Alderaan_Energy_ADE_Schedule_Library.gml Energy ADE 3.0: • ScheduleLibrary, DualValueSchedule, AtomicSchedule, CompositeSchedule, ScheduleComponent • TypicalValuesTimeSeries This file contains a collection of different schedules contained within a *ScheduleLibrary* object. *AtomicSchedule* objects are connected to *TypicalValueTimeSeries* objects. This dataset uses classes from the Energy ADE 3.0 Schedules and TimeSeries modules.	
Alderaan_Energy_ADE_Core_AncillaryBuildings.gml CityGML: • Building • CityObjectGroup Energy ADE 3.0: • ADE _CityObject This file contains a group of 11 buildings modelled in LoD0 and LoD1. Each building is also modelled through a reference point. The *CityObjectGroup* containing them is modelled through a reference point too, and has a multisurface geometry representing the extents of the area (not shown here). This dataset uses classes from the Energy ADE 3.0 Core module.	

Alderaan_Energy_ADE_All.gml This file merges and integrates together information from all previous files. For visual clarity reasons, *CityObjectGroup* and *UrbanFunctionArea* objects are included in the file, but not visualised in the screenshot on the right. This dataset uses classes from all Energy ADE 3.0 modules.	
Alderaan_ADE_All_no_DTM.gml This file is the same as the previous one, with the only difference that it does not contain any *Relief* objects. For visual clarity reasons, *CityObjectGroup* and *UrbanFunctionArea* objects are included in the file, but not visualised in the screenshot on the right. This dataset uses classes from all Energy ADE 3.0 modules.	

7 Database implementation

The 3D City Database (or, in short: 3DCityDB) is a free and open-source solution to store, represent, and manage CityGML-compliant semantic 3D city models using a spatial relational database. Different version of the 3DCityDB are currently available. The 3DCityDB 4.x supports CityGML 1.0 and 2.0 (but no CityGML 3.0) and comes with a set of tools consisting mainly of an Importer/Exporter, a Graphical User Interface, support for ADEs, and other tools, e.g. to export city models to KML/Kollada or glTF formats for further visualisation in virtual-globe applications such as Google Earth or CesiumJS.

The latest release of the 3DCityDB, version 5.0, has been released at the beginning of 2025 and adds support for CityGML 3.0. The new 3DCityDB comes with a completely redesigned database schema, as well as a new (GUI-less) set of tools to import and export data to and from the database. Noteworthy, at the time of writing, the database does **not** offer support for ADEs, yet. Besides, there are currently no available examples of ADEs (for any version of CityGML) implemented for the latest version of the 3DCityDB.

Regardless of the version, the 3DCityDB is available for PostgreSQL/PostGIS. Considering that the Energy ADE 3.0 has been developed first for CityGML 2.0, the decision on which version and platform to work for is rather natural: 3DCityDB 4.x due to its support for ADEs, and PostgreSQL/PostGIS due to its free and open-source nature.

7.1 Installation

Installation of the Energy ADE 3.0 database schema is rather straightforward and can be carried out using the ADE Manager from within the 3DCityDB Importer/Exporter GUI.

The current version of the Energy ADE 3.0 plugin for the 3DCityDB requires:

- **3DCityDB version 4.4.2**or higher (please note: 3DCityDB version 5.x is NOT supported!)
- **3DCityDB Importer/Exporter 5.5.1** or higher
- **ADE Manager version 2.3.2** or higher.

All required software conveniently downloaded and installed using the **3DCityDB Suite 2024.0.3** package (or a more recent version) available from the 3DCityDB GitHub repository[26].

As shown in Figure 33, the user simply has to browse to the **\energy-ade3** folder containing the ADE extension package, i.e. the folder containing all resources required by ADE Manager to install and set up the database to support the Energy ADE 3.0.

[26] https://github.com/3dcitydb/3dcitydb-suite/releases/tag/v2024.0.3

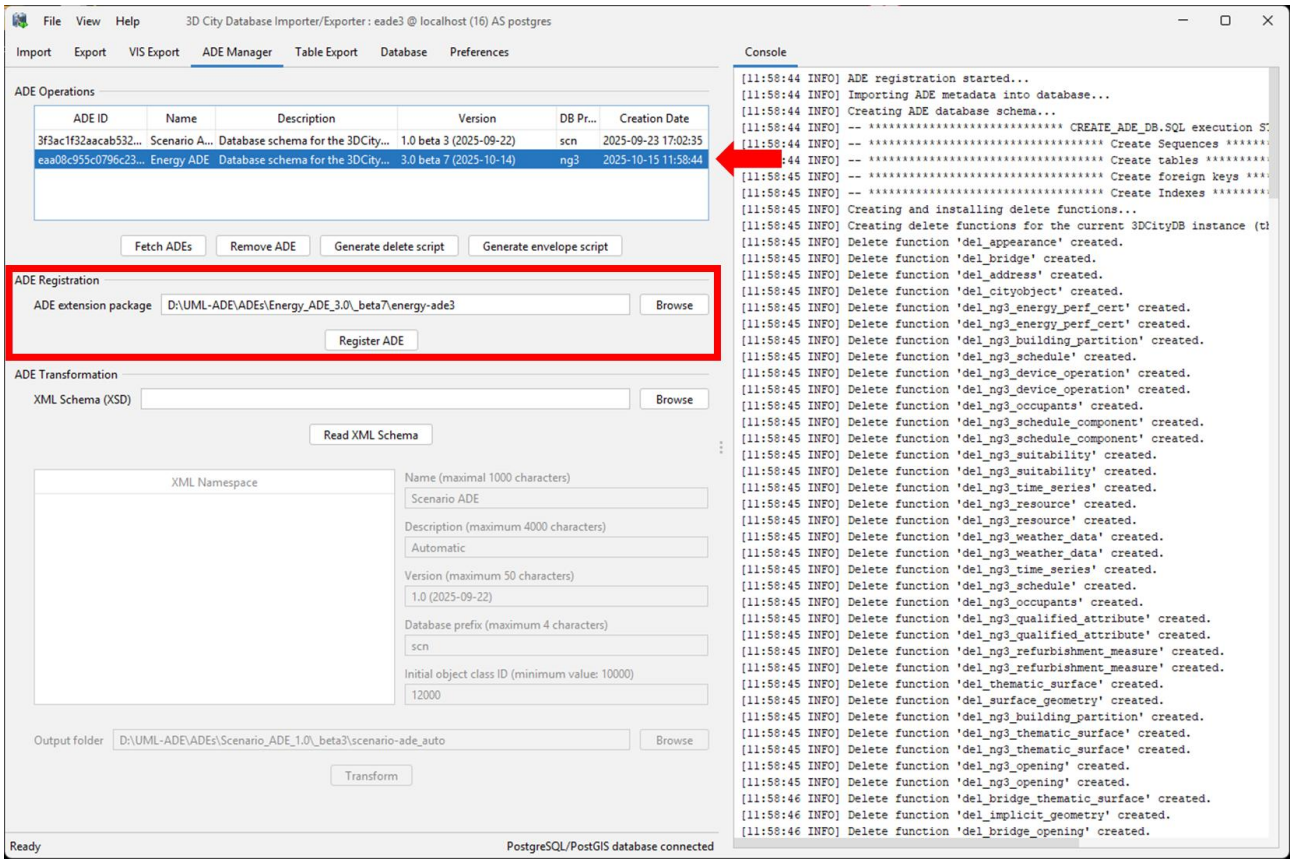


Figure 33. Screenshot of the ADE Manager of the 3DCityDB Importer/Exporter with the ADE extension package to be installed. Upon installation, the ADE metadata will appear in the list of registered ADEs.

7.2 Uninstallation

Uninstallation of the Energy ADE 3.0 database schema can be carried out in a similar way as the installation. After connecting to the database and setting the current *citydb* schema, it suffices to first load the list of the installed ADEs pressing the button "Fetch ADEs", then choose to ADE and press the “Remove ADE” button from within the ADE Manager, as shown in Figure 34. Please note that all Energy ADE contents will be deleted automatically, while "standard" CityGML data will be preserved. If you want to manually delete all Energy ADE 3.0 data without uninstalling the Energy ADE, please refer to the next section 7.3.

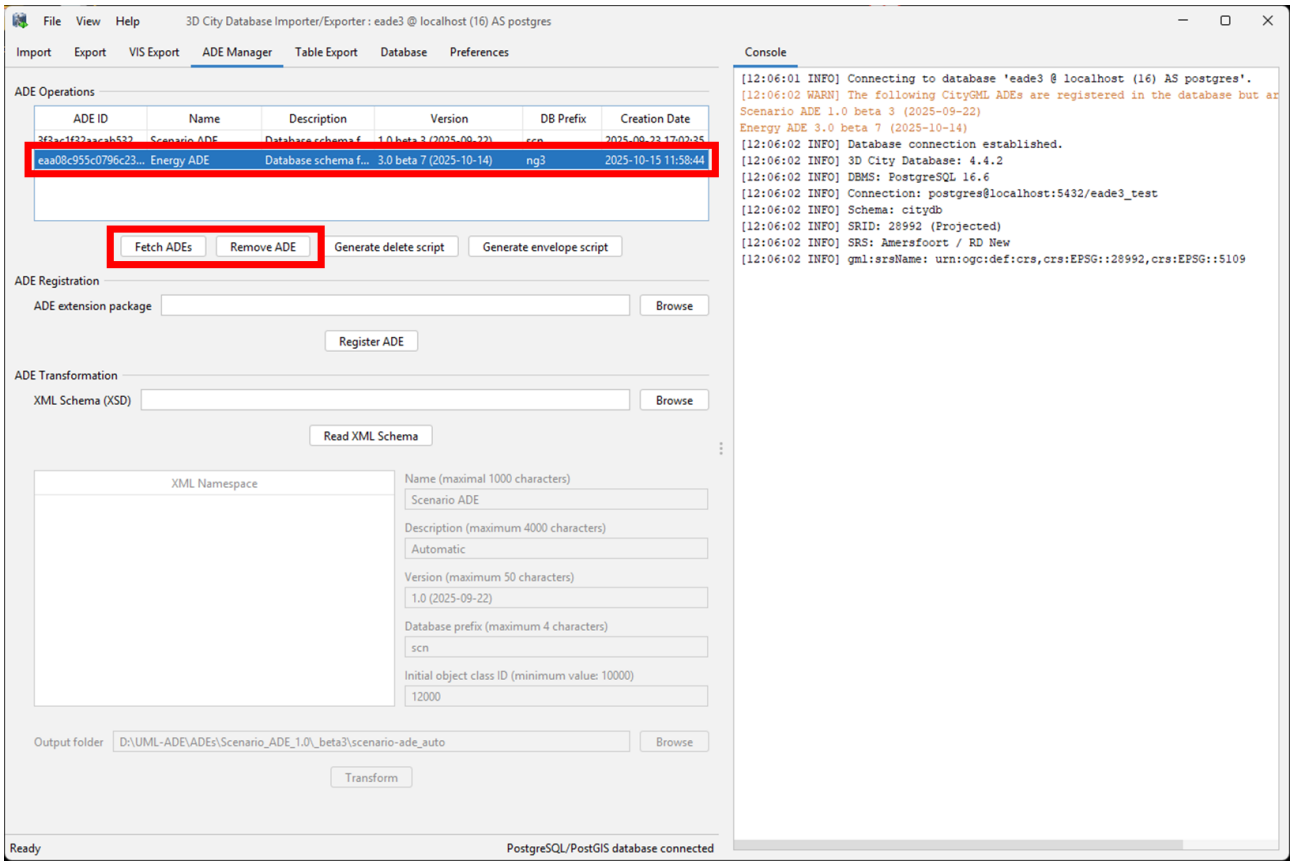


Figure 34. Screenshot of the ADE Manager of the 3DCityDB Importer/Exporter with the selected ADE to be removed. Upon uninstallation, all ADE data will be removed before dropping all ADE tables and other related entities.

7.3 Manual deletion of *all* Energy ADE 3.0 data

Specific Energy ADE 3.0 data can be deleted using the delete functions listed in Table 4 of section 7.6.7. However, a faster alternative to delete *all* Energy ADE data *at once* (and to keep the "standard" CityGML data) is to run the code block in Listing 1. It is an omni-comprehensive set of SQL statements that will delete any Energy ADE 3.0 data from the chosen *citydb* schema (the default value, highlighted in yellow, can be edited by the user). Alternatively, the code block can be customised depending on specific needs.

The code block can be copy-pasted into a SQL command window and directly run, e.g. from PgAdmin.

```
--SQL code block to manually delete all Energy ADE 3.0 from the chosen citydb schema
DO $$
DECLARE cdb_schema varchar := 'citydb';
BEGIN
IF cdb_schema IS NULL OR cdb_schema = '' THEN RAISE EXCEPTION 'You must provide a value for the 3DCityDB name'; END IF;
-- Delete Energy ADE 3.0 Data types
EXECUTE format('SELECT %I.del_ng3_suitability(id) FROM %I.ng3_suitability', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_optical_property(id) FROM %I.ng3_optical_property;', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_qualified_attribute(id) FROM %I.ng3_qualified_attribute;', cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Time Series module
EXECUTE format('SELECT %I.del_ng3_time_series(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''RegularTimeSeries'', ''RegularTimeSeriesFile'', ''TypicalValuesTimeSeries'', ''TypicalValuesTimeSeriesFile'', ''SensorConnection'')', cdb_schema, cdb_schema,
cdb_schema);
-- Delete Energy ADE 3.0 Schedules module
EXECUTE format('SELECT %I.del_ng3_schedule(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''AtomicSchedule'', ''DualValueSchedule'', ''CompositeSchedule'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_schedule_component(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''ScheduleComponent'')',
cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_library(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''ScheduleLibrary'')', cdb_schema,
cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Layered Construction module
EXECUTE format('SELECT %I.del_ng3_material(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''Gas'', ''SolidMaterial'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_layer(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''Layer'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_layered_construction(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''LayeredConstruction'', ''ReverseLayeredConstruction'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_library(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''MaterialLibrary'', ''LayeredConstructionLibrary'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Resource module
EXECUTE format('SELECT %I.del_ng3_resource(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''Energy'', ''Water'', ''Waste'', ''Food'', ''UrbanSpace'', ''ConstructionMaterial'', ''OtherResource'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Weather station module
EXECUTE format('SELECT %I.del_cityobject(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''WeatherStation'')', cdb_schema,
cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 A bit of Core module
EXECUTE format('SELECT %I.del_ng3_ctyobj_relation(id) FROM %I.ng3_ctyobj_relation', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_weather_data(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''WeatherData'')', cdb_schema,
cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_utl_ntw_connection(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''UtilityNetworkConnection'')',
cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_energy_perf_cert(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''EnergyPerformanceCertificate'')',
cdb_schema, cdb_schema, cdb_schema);
```

```
EXECUTE format('SELECT %I.del_ng3_refurbishment_measure(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''RefurbishmentMeasure'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Urban Function Area module
EXECUTE format('SELECT %I.del_ng3_urban_function_area(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''UrbanFunctionArea'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Devices module
EXECUTE format('SELECT %I.del_ng3_storage_device(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''ThermalStorageDevice'', ''ElectricalStorageDevice'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_solar_collector(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''GenericSolarCollector'', ''PhotovoltaicCollector'', ''SolarThermalCollector'', ''PhotovoltaicThermalCollector'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_device(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''Boiler'', ''LightingDevice'', ''GenericDevice'', ''GenericElectricalDevice'', ''MovableShadingDevice'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Delete the PartyWalls that may be used for Buildings and not for ThermalZones
EXECUTE format('SELECT %I.del_thematic_surface(co.id) FROM %I.thematic_surface AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''PartyWallSurface'')
        WHERE co.building_id IS NOT NULL', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Occupancy module
EXECUTE format('SELECT %I.del_ng3_occupants(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''Occupants'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_building_partition(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''BuildingUnit'')', cdb_schema, cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_building_partition(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''UsageZone'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 Building physics module
EXECUTE format('SELECT %I.del_thematic_surface(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id)
        WHERE o.classname IN (''PartyWallSurface'', ''BasementCeilingSurface'', ''IntermediateFloorSurface'', ''AttiFloorSurface'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 dependent openings used for thermal boundaries will be dropped automatically on cascade
EXECUTE format('SELECT %I.del_ng3_building_partition(co.id) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname=''ThermalZone'')', cdb_schema, cdb_schema, cdb_schema);
-- Delete Energy ADE 3.0 A bit of Core module: Delete only ADE data from the ng3 tables
EXECUTE format('SELECT %I.del_ng3_opening(id) FROM %I.ng3_opening', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_thematic_surface(id) FROM %I.ng3_thematic_surface', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_building(id) FROM %I.ng3_building', cdb_schema, cdb_schema);
EXECUTE format('SELECT %I.del_ng3_cityobject(id) FROM %I.ng3_cityobject', cdb_schema, cdb_schema);
RAISE NOTICE 'SUCCESS! Energy ADE 3.0 contents deleted from citydb schema "%"', cdb_schema;
END $$;
```

Listing 1. SQL code block to delete at once all Energy ADE 3.0 data from the selected 3DCityDB schema (here, highlighted in yellow: citydb). Please note: no CityGML data will be deleted, only Energy ADE 3.0 data.

7.4 Known limitations

7.4.1 The Importer/Exporter GUI fails to compute/update bounding boxes

Upon import of Energy ADE 3.0 data, the Importer/Exporter GUI allows to compute/update the envelope of the data contained in the database. This operation can be carried out by simply clicking the "Create missing" or "Recreate all" buttons in the Database tab, as shown in Figure 35.

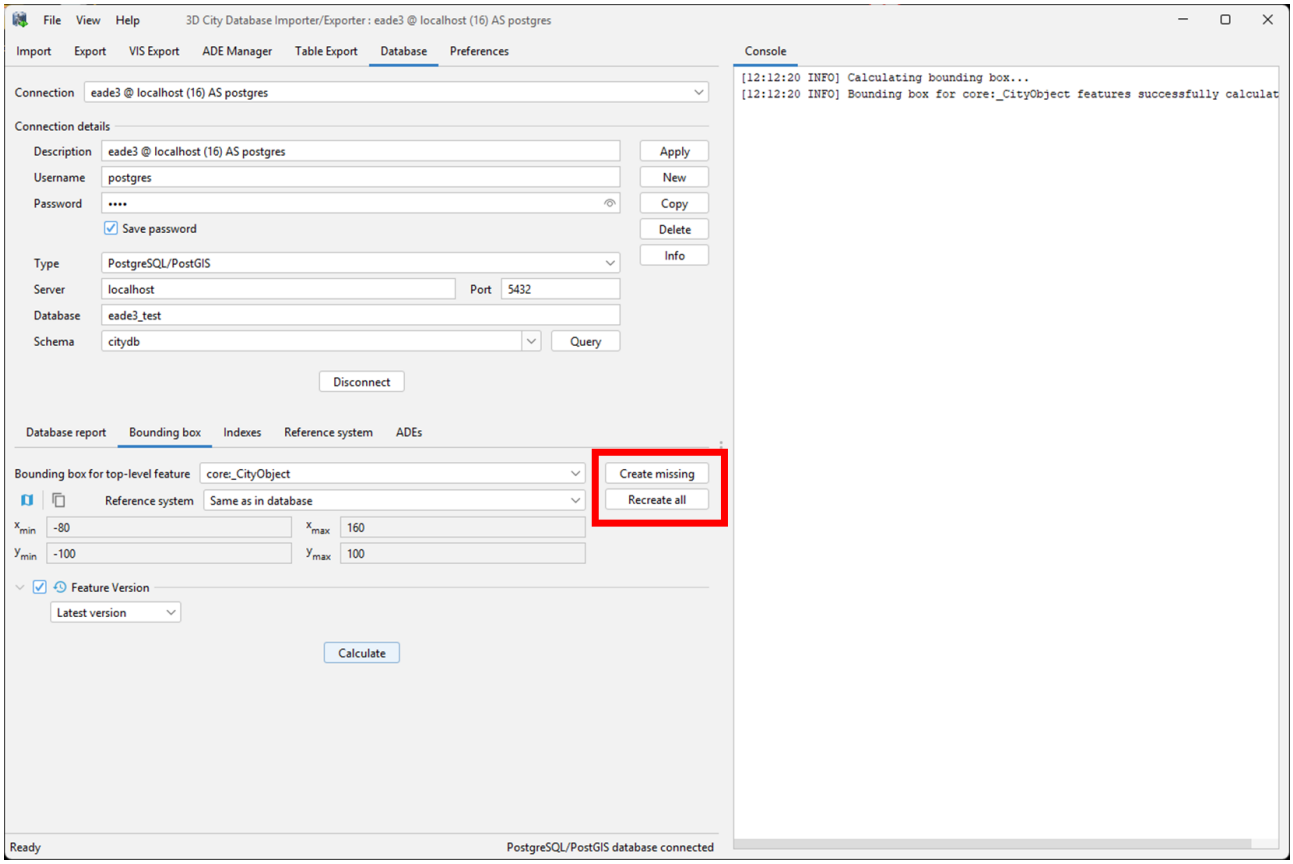


Figure 35. Screenshot of the 3DCityDB Importer/Exporter, Database tab, showing how compute/update the bounding boxes of the database objects.

However, at the moment, it is not possible to compute the 3D bounding box of Energy ADE 3.0 top-level features, such as *WeatherStation*, *UrbanFunctionArea*, *ScheduleLibrary*, *MaterialLibrary* and *LayeredConstructionLibrary* using the above mentioned GUI buttons. This functionality will be made available once the corresponding Java libraries will be implemented (see section 9.2).

The current workaround consists in executing the SQL code block provided in Listing 2, for example in a SQL console in PgAdmin. The script can be adapted to be run with any 3DCityDB schema. The default value is *citydb*.

```
--SQL code block to manually compute and update the envelope in the chosen citydb schema
DO $$
DECLARE cdb_schema varchar := 'citydb';
BEGIN
```

```
IF cdb_schema IS NULL OR cdb_schema = '' THEN RAISE EXCEPTION 'You must provide a value for the 3DCityDB name'; END IF;
-- Compute 3D bounding box and update CITYOBJECT table for:
-- Librariy objects
EXECUTE format('SELECT %I.env_ng3_library(id, 1) FROM %I.ng3_library', cdb_schema, cdb_schema);
-- UrbanFunctionArea objects
EXECUTE format('SELECT %I.env_cityobjectgroup(id, 1) FROM %I.ng3_urban_function_area', cdb_schema, cdb_schema);
-- WeatherStation objects
EXECUTE format('SELECT %I.env_cityobject(co.id, 1) FROM %I.cityobject AS co JOIN %I.objectclass AS o ON (co.objectclass_id = o.id AND o.classname = ''WeatherStation'') ', cdb_schema, cdb_schema, cdb_schema);
RAISE NOTICE 'SUCCESS! Computed 3D bounding box and updated CITYOBJECT table for Energy ADE 3.0 top-level CityObjects in citydb schema "%"', cdb_schema;
END $$;
```

Listing 2. SQL code block to compute the 3D bounding boxes and update the CITYOBJECT table for the Energy ADE 3.0 top-level city objects in the selected 3DCityDB schema (here, highlighted in yellow: citydb).

7.5 Derivation of the DDL scripts

7.5.1 Automatic process

Once the XSD file of an ADE is finalized, as in the case of the Energy ADE 3.0, it can be used to derive the database objects needed to store data according to the corresponding SQL encoding. For this purpose, the Importer/Exporter for the 3DCityDB 4.x comes with a plugin, the ADE Manager, which can automatically derive the Data Definition Language[27] (DDL) scripts required to set up all database objects to support the ADE, namely: tables, constraints, indices, sequences, and several stored procedures, among which, for example, those to delete data and to compute and store the bounding boxes of city objects. While providing a detailed description of the ADE Manager is beyond the scope of this document, it suffices to say that it stems from research work carried out at TU Munich in the context of a PhD thesis (Yao, 2020). More details can be found in the 3DCityDB online manual[28].

From the user point of view, the process is rather simple and is shown in Figure 36. It is only necessary to have the plugin read the XSD file, to set few metadata parameters, and chose the destination output folder that will contain the generated scripts. In the case of the Energy ADE 3.0, as shown in Figure 36, the process results with the generation of 59 new tables (and several additional database objects). All new database objects are prefixed with a **database prefix**. In the case of the Energy ADE 3.0, it is "**ng3**".

[27] https://en.wikipedia.org/wiki/Data_definition_language

[28] https://3dcitydb-docs.readthedocs.io/en/release-v4.2.3/impexp/plugins/ade-manager.html

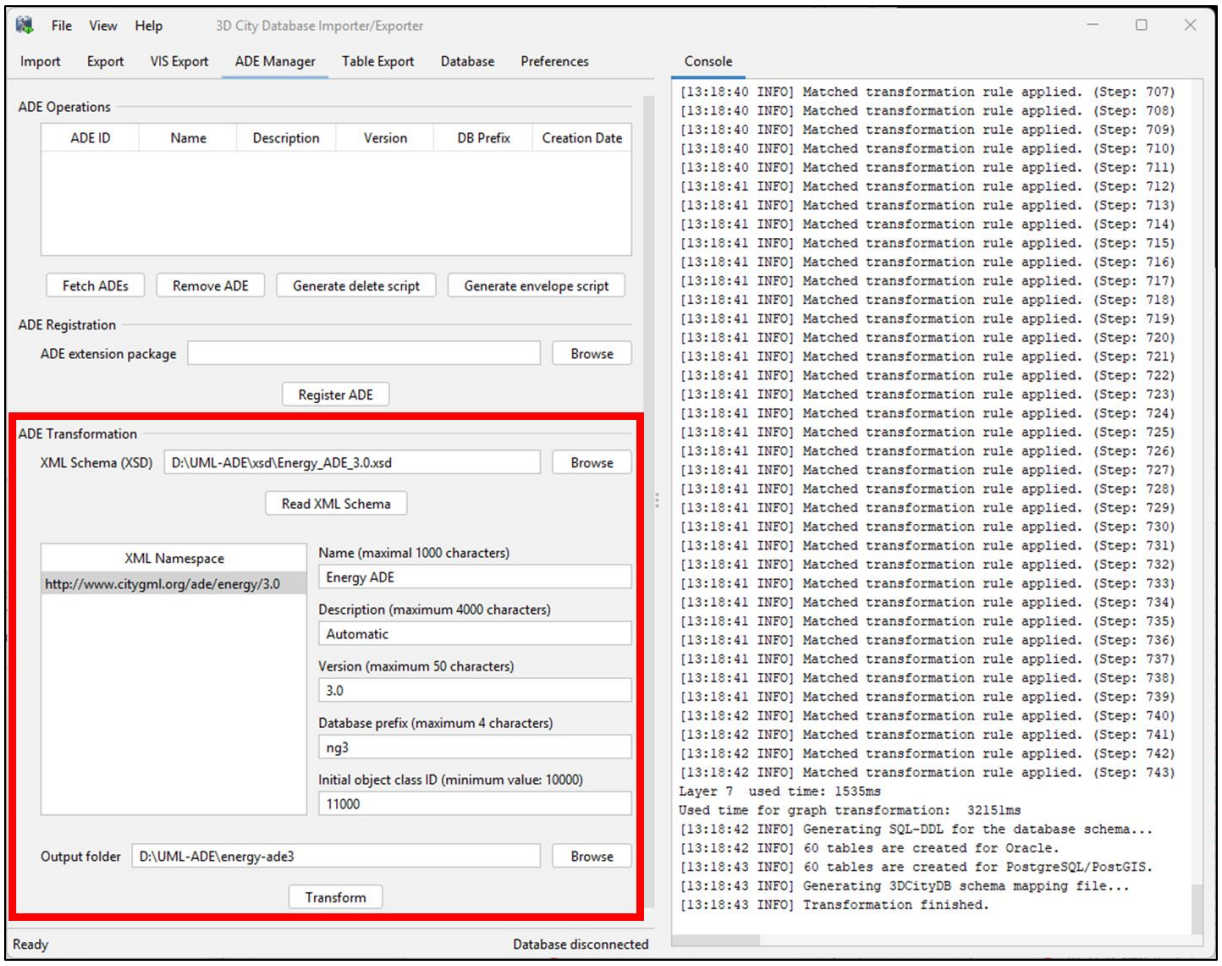


Figure 36. Example of automatic generation of the DDL scripts for the Energy ADE 3.0 using the ADE Manager plugin provided with the 3DCityDB Importer/Exporter.

7.5.2 Manual refinement

The DDL scripts generated in the previous step are already sufficient to set up the 3DCityDB and support the Energy ADE 3.0. The installation procedure is briefly described in section 7.1. Upon installation, the new ADE tables, constraints, sequences, indices, and function are installed automatically.

The fully automatic mapping process comes however with few disadvantages. Two main types of disadvantages are mentioned here:

1) **Number of new tables**. Depending on the underlying data models, the automatic mapping rules may lead to the creation of many tables
2) **Names of tables, columns, etc**. The mapping rules take care of automatically converting the names of elements in the XSD file (corresponding to names of classes, attributes and relations in the UML diagram) to database objects. If needed, some names are automatically trimmed to reduce their length for database portability reasons. The resulting names of tables, columns, etc., may be however sometimes unclear.

Overcoming both types of shortcoming is possible but requires some additional manual tuning. In the case of the Energy ADE 3.0, additional effort has been put into this optimisation phase, so that, for example, the number of tables has been reduced from 59 to 29. At the same time, all table and column names have been checked and renamed, if necessary. The resulting DDL scripts have been adapted accordingly. Tests have been carried out in order to ensure that

all other automatically generated objects (e.g. the delete and envelope functions) still work as expected. For example, the automatic mapping of the Resources module generates 8 new tables (NG3_RESOURCE_, NG3_CONSTRUCTIONMATERIAL, NG3_ENERGY, NG3_FOOD, NG3_OTHERRESOURCE, NG3_URBANSPACE, NG3_WATER, NG3_WASTE), as shown in Figure 37.

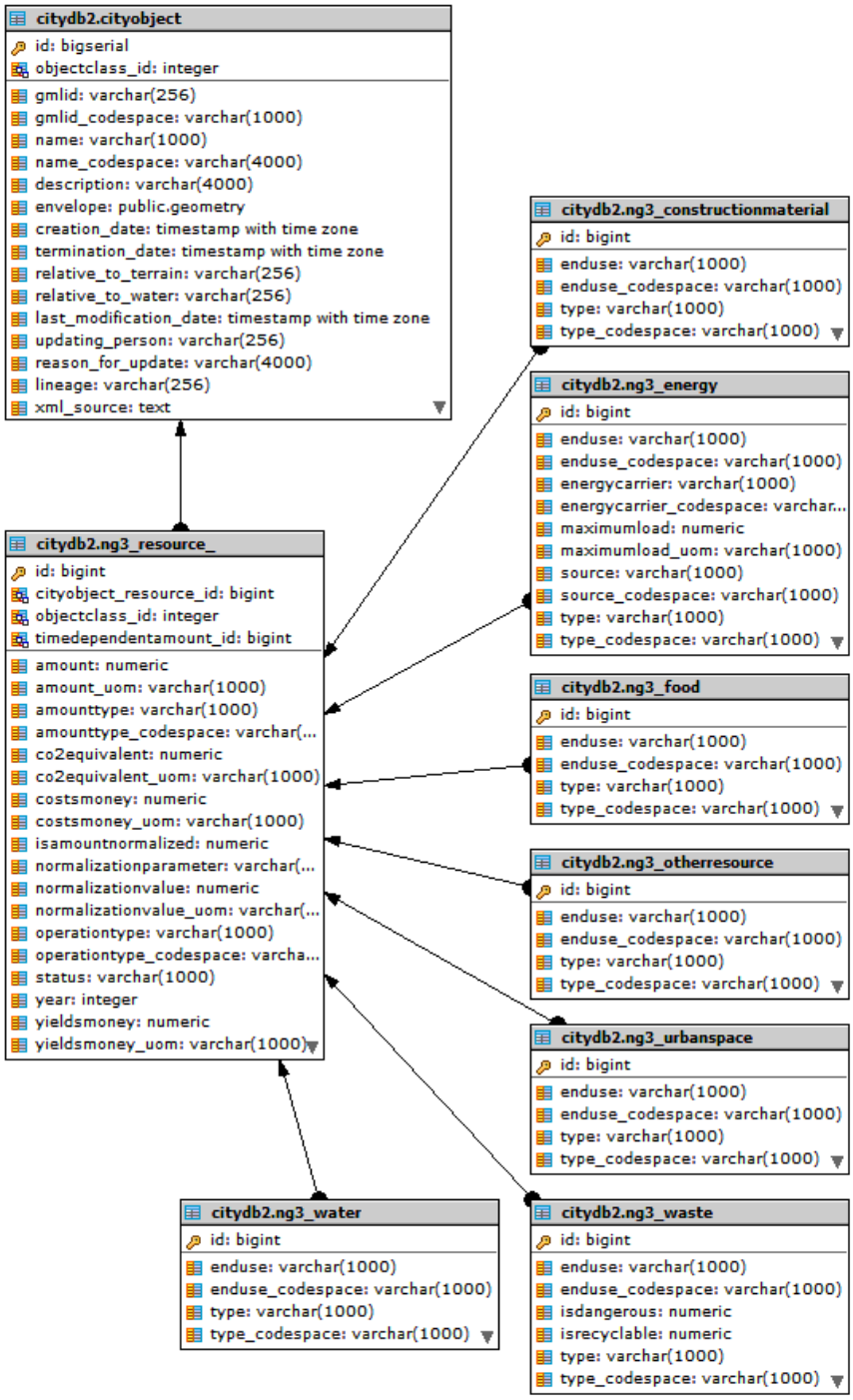


Figure 37. Entity-Relationship diagram showing the results from the automatic mapping of the Energy ADE 3.0 Resources module (the linked table NG3_TIME_SERIES is here omitted for the sake of better readability).

The manual refinement reduces the number to just one table (NG3_RESOURCE). Column names have been slightly changed too. The result is shown in Figure 38.

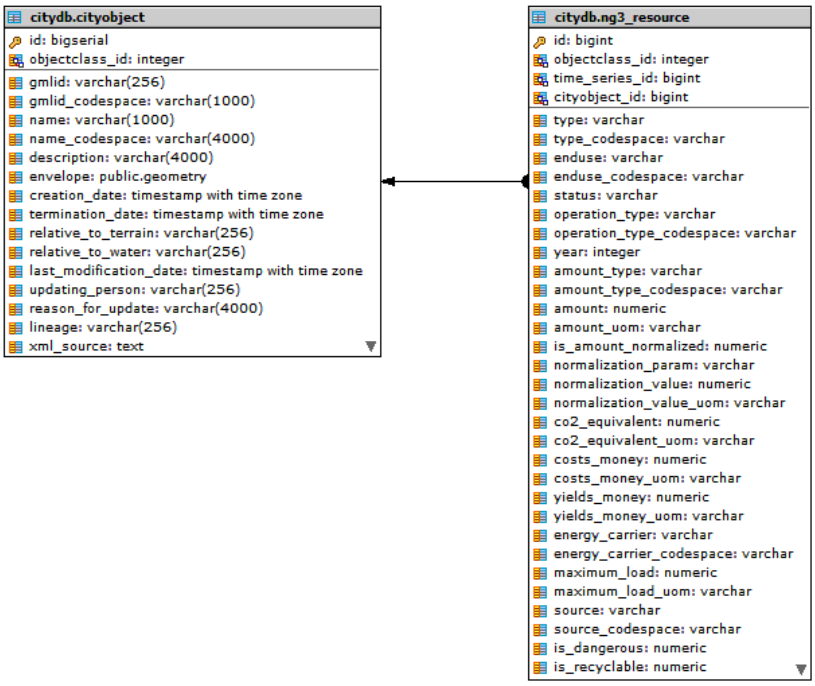


Figure 38. Entity-Relationship diagram showing the results from the manual refined mapping of the Energy ADE 3.0 Resources module (the linked table NG3_TIME_SERIES is here omitted for better readability).

It is important to highlight that the database design rules adopted in the manual refinement step and previously shown with the Resources module have been used several times: wherever possible, multiple classes have been mapped to the same table, and the resulting table contains the sum of all attributes of the corresponding classes. This means that, depending on the class, some columns will be written while other may remain empty. This design pattern is actually not new to the 3DCityDB. For example, all CityGML *generic attributes* are mapped to only one table, namely the CITYOBJECT_GENERICATTRIB table. Despite the disadvantage of some waste of storage space due to columns that may remain empty, the advantages consist in less joins, less indices, less constraints, less stored procedures, and, overall, in a more compact database structure. More details about the specific mapping decisions will be provided in section 7.7.

7.6 From OO- to ER-model: General mapping rules

Regardless of the mapping strategies mentioned before (fully automatic or manually refined), some overarching rules apply to the way the ADE classes are mapped to the 3DCityDB 4.x schema. This section briefly introduces them as they are relevant in case a user wants to interact directly with the database.

In general, and in line with the mapping rules adopted by the 3DCityDB for "standard" CityGML tables, one or more ADE classes can be mapped to one single table. The name of the table is identical to the class name in most of the cases; a leading underscore indicating an abstract class is left out. Every ADE table is prefixed with the database prefix (e.g. “ng3” for the Energy ADE 3.0). ADE classes may be combined into a single table according to the class relations. If

this is the case, very often the respective table has a column named *objectclass_id*, which contains a foreign key to table OBJECTCLASS where information about all classes (both "standard" CityGML and ADE classes) is available.
When it comes to class properties, they are mapped to table columns with identical (or very similar) names. For every attribute including measure information an additional "_uom"-suffixed column is provided to specify the unit of measurement. For example, in the Energy ADE 3.0, attributes of class ADE *_BoundarySurface* that are of type Length (e.g. *thickness*) or Area (e.g. *opaqueSurfaceArea*), are mapped to columns *thickness*, *thickness_uom*, *opaque_surf_area* and *opaque_surf_area_uom*, respectively. Properties of type gml:Code or associated to a CodeList are mapped to two columns too. The first one contains the value of the attribute, the second one has the same name, but is suffixed with "_codespace" and it is meant to store the (optional) code space of that attribute.
Please note that in the following sections not all attributes of every single table will be explained in detail, especially those for which the mapping is rather self-explanatory.

Depending on the ADE class of the Energy ADE 3.0, an easy classification of the respective set of mapping rules can be done by looking at *type* of a class and its *stereotype*. In the Energy ADE 3.0, three main stereotypes are used: «ADEElement», «featureType», and «dataType». Classes belonging to other stereotypes (e.g. «enumeration», or «codeList») will be not discussed here. In the UML diagrams, the stereotype is generally written above the name of the class and between guillemots (i.e. «…»), as shown in the examples of Figure 39.

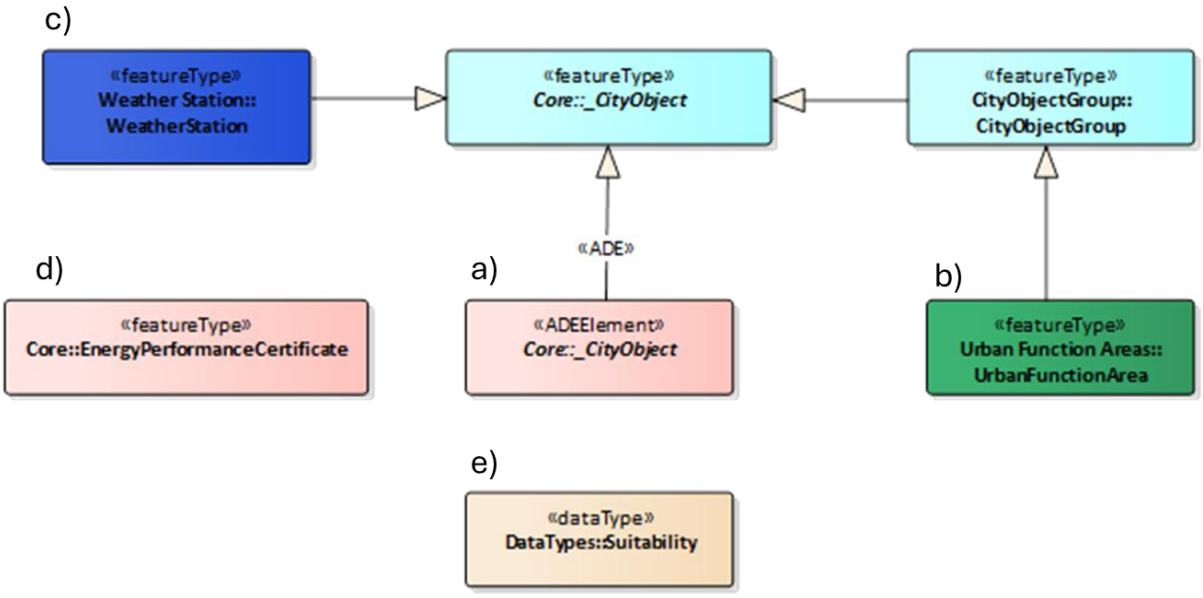


Figure 39. UML diagram showing some classes, as well as their stereotype above the class name, written within guillemots (i.e. « »).

7.6.1 Mapping of extended city objects

ADE classes that extend existing CityGML classes by means of the so-called ADE-hook mechanism are easily recognizable because of the «ADEElement» stereotype written in the UML diagram above their names. In Figure 39 an example is provided by the ADE *_CityObject* class identified with letter a). These ADE classes are mapped to the database as follows:

An additional table, prefixed with the database prefix, is joined to the existing table containing the original class attributes. Save for the database prefix, the name of the table is the same as the one it is linked to. For example: in the Energy ADE 3.0, the ADE *_CityObject* class is mapped to table NG3_CITYOBJECT. The *id* column serves as primary key for table NG3_CITYOBJECT and, at the same time, as foreign key to table CITYOBJECT

- Properties with multiplicity higher than 1 (no matter whether simple or complex) are generally mapped to external, linked tables. For example: In the Energy ADE 3.0, the *suitability* property of class ADE *_CityObject* is mapped to the additional, linked table NG3_SUITABILITY
- Geometries are stored in their own columns in the same table (for example: reference point of class ADE *_CityObject*), unless they contain multisurface or solid geometry data. In this case, the usual 3DCityDB mapping rules apply: such geometries are decomposed into polygons and stored in table SURFACE_GEOMETRY. Further details can be found in the 3DCityDB online documentation[29].

[29] https://3dcitydb-docs.readthedocs.io/en/latest/3dcitydb/uml/geometry.html#citydb-geometric-topological-model

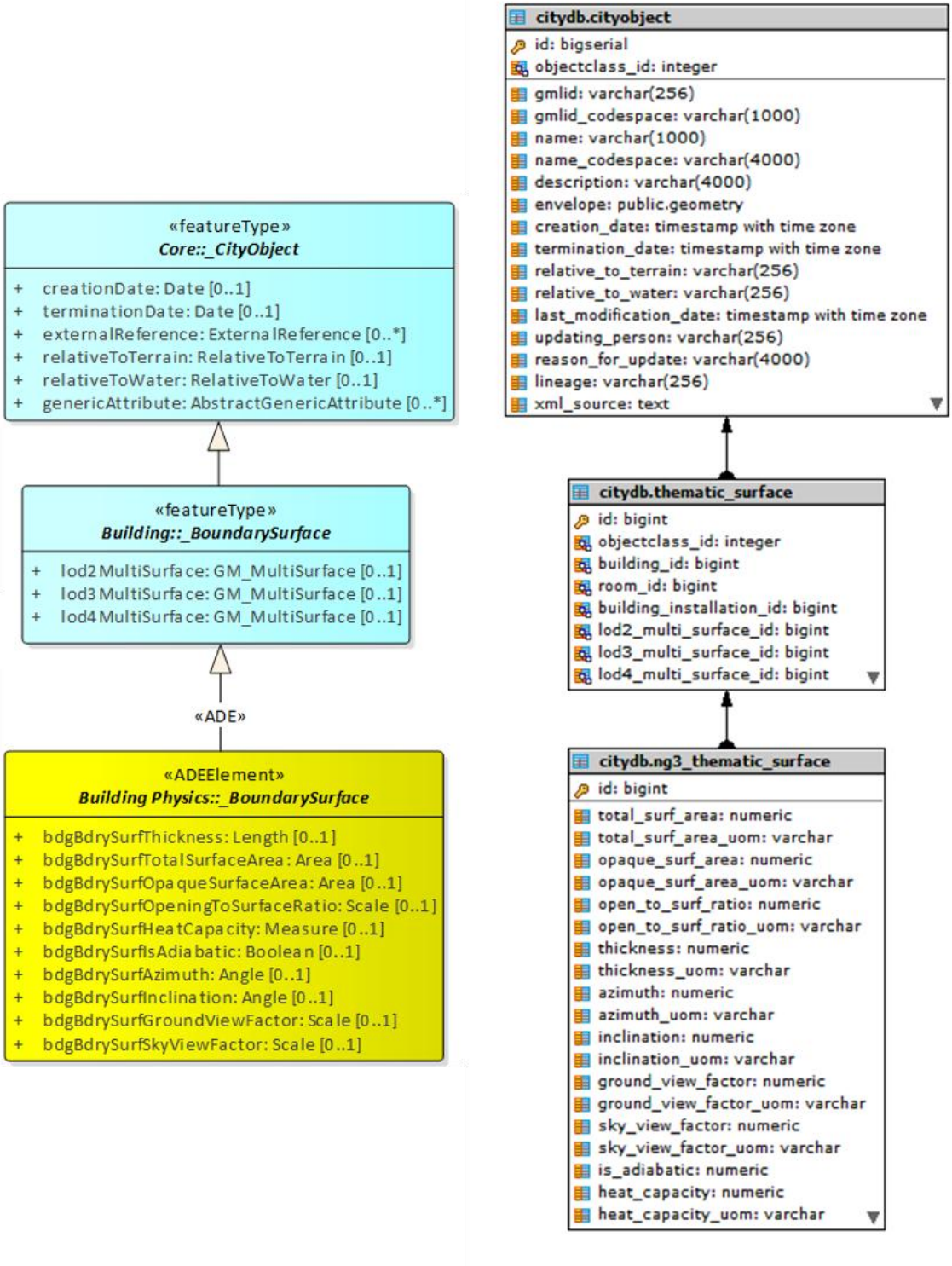


Figure 40. Mapping example of CityGML class *_BoundarySurface* extended by means of the ADE-hook mechanism. Please note the «ADEElement» stereotype written on top of the ADE class.

7.6.2 Mapping of new city objects derived from existing city objects

A new ADE class can be derived by means of specialization from an existing CityGML class (itself derived from *_CityObject*). For such classes, the stereotype in the UML diagrams is «featureType». In Figure 39 an example is provided by class *UrbanFunctionArea*, identified with letter b), which further specializes CityGML class *CityObjectGroup*.

The mapping rules for this type of classes are similar to those described in the previous section 7.6.1. All properties inherited from parent CityGML classes are mapped to the tables corresponding to the parent CityGML class. The new attributes are stored in a new table, prefixed with the database prefix, and linked to the last table of the parent class. For example, in the Energy ADE 3.0, all inherited properties up to class *CityObjectGroup* are stored in tables CITYOJBECT and CITYOBJECTGROUP. The properties of *UrbanFunctionArea* (*type* and *code*) are stored in table NG3_URBAN_FUNCTION_AREA, as shown in Figure 41.

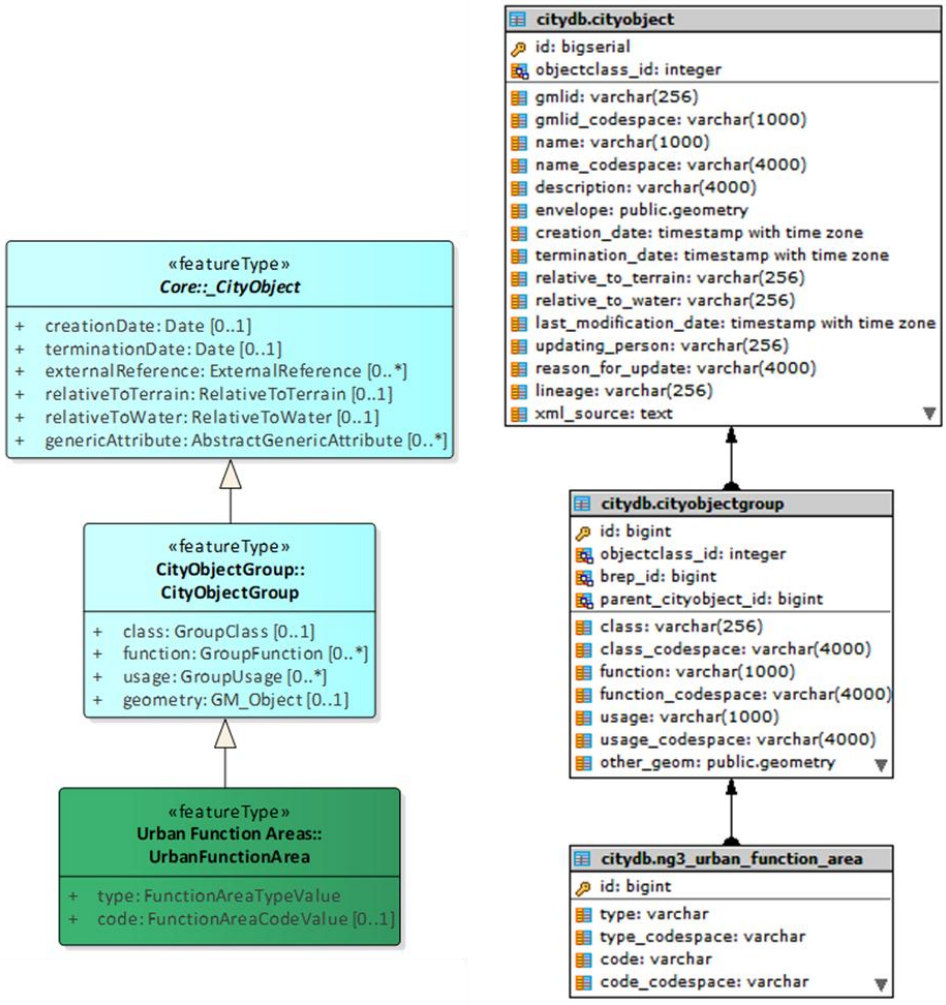


Figure 41. Mapping example of Energy ADE 3.0 class *UrbanFunctionArea*, derived from CityGML class *CityObjectGroup*.

7.6.3 Mapping of new city objects

A new ADE class can be derived by means of specialization directly from the CityGML class *_CityObject*. For such classes, the stereotype in the UML diagrams is «featureType». In Figure 39. An example is provided by class *WeatherStation*, identified with letter c).

The mapping rules for this type of classes follow the same rules as for any CityGML analogous class derived from *_CityObject*. Starting from table CITYOBJECT, additional attributes are stored, if available, in additional linked tables. In the case of ADE classes, the tables are prefixed with the database prefix. Geometries are stored either inline in the same table or, in case of

solids and multisurface geometries, in table SURFACE_GEOMETRY and then referenced by means of foreign keys.
For example, in case of the Energy ADE 3.0, classes *MaterialLibrary* and *WeatherStation* fall within this set of mapping rules: class *MaterialLibrary* is mapped to tables CITYOBJECT + NG3_LIBRARY, while for class *WeatherStation* only table CITYOBJECT is required. Figure 42 shows how class *MaterialLibrary* is mapped to the corresponding tables.

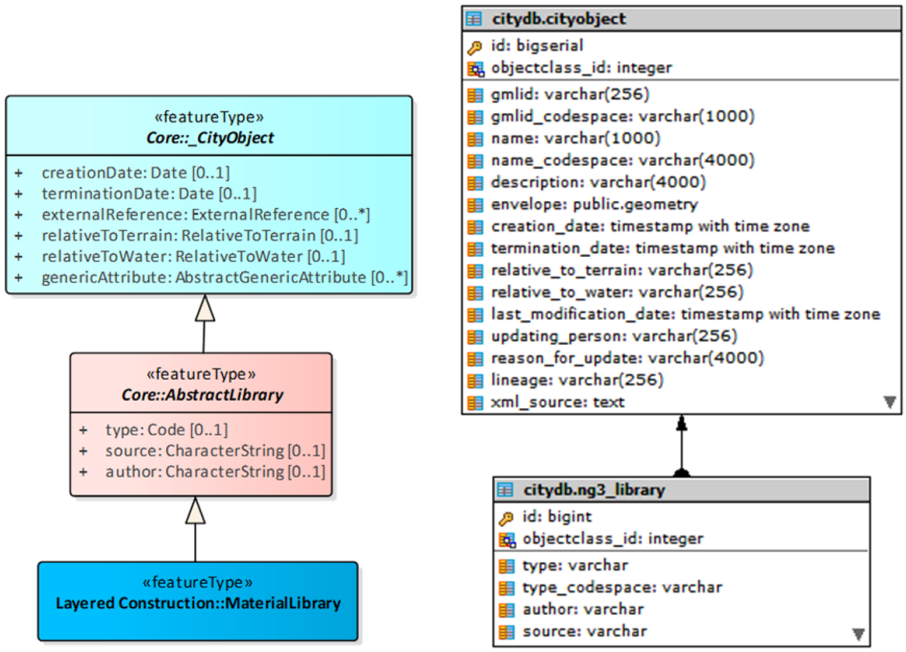


Figure 42. Mapping example of Energy ADE 3.0 class *MaterialLibrary*.

7.6.4 Mapping of features

New ADE classes can be derived by specialisation directly from GML abstract base class *_Feature*. In this case, such classes do not inherit any properties from CityGML class *_CityObject*, as they are derived from its parent class instead. In Figure 39 an example is provided by the class *EnergyPerformanceCertificate*, identified with letter d).
The mapping rules for this type of classes are similar to those described in the previous section 7.6.4. All properties inherited from parent *_Feature* class are mapped to table CITYOBJECT. However, as a result of not being a subclass of *_CityObject*, some fields in table CITYOBJECT remain unused. The specific class properties are stored in a new table, prefixed with the database prefix, and linked to CITYOBJECT. For example: In the Energy ADE 3.0, class *EnergyPerformanceCertificate* is mapped to tables CITYOJBECT + NG3_ENERGY_PERF_CERT, as shown in Figure 43.

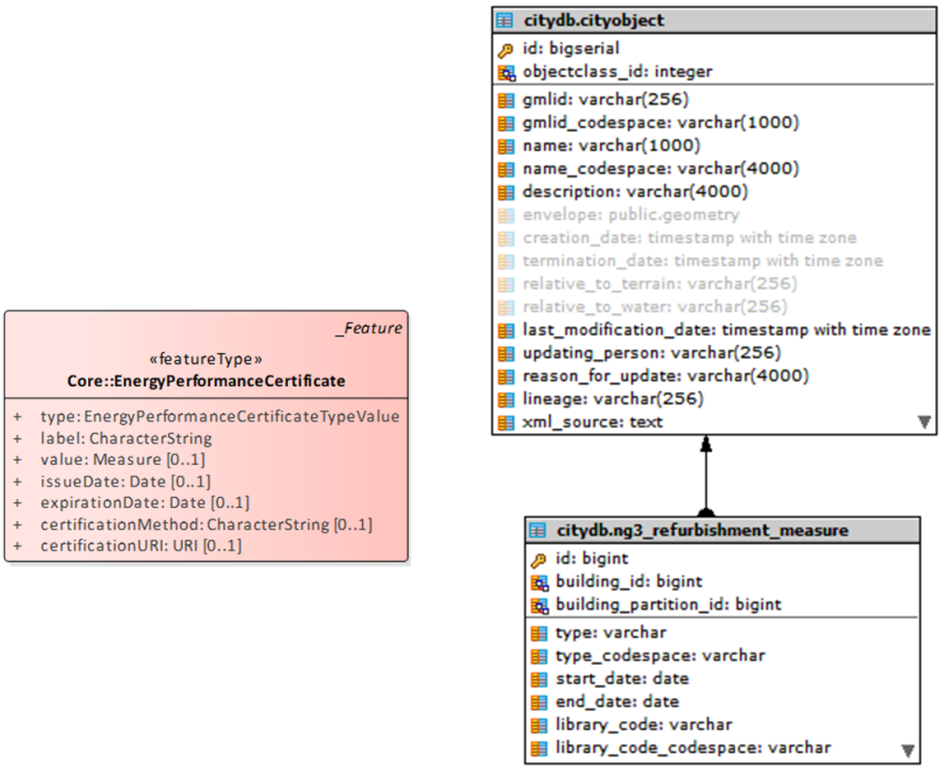


Figure 43. Mapping example of Energy ADE 3.0 class *EnergyPerformanceCertificate*. Please note that not all fields of table CITYOBJECT are used, as class *EnergyPerformanceCertificate* is derived from GML class *_Feature*, but not from CityGML class *_CityObject*.

7.6.5 Mapping of data types

An example of a class belonging to stereotype «dataType» is presented in Figure 39 and identified with letter e). In the Energy ADE 3.0 only few classes belong to this group, namely those derived from classes *AbstractQualifiedAttribute*, *AbstractOpticalProperty*, and class *Suitability*. In each case, they are mapped to the respective tables: NG3_QUALIFIED_ATTRIBUTE, NG3_OPTICAL_PROPERTY, and NG3_SUITABILITY. Each table has its own sequence, and some fields are used for foreign keys to the respective parent objects. Examples are provided in Figure 44.

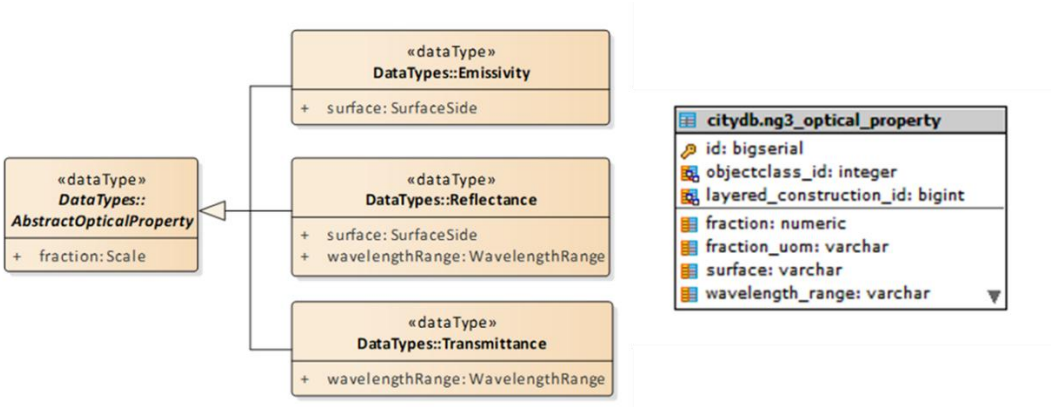


Figure 44. Mapping example of Energy ADE 3.0 classes belonging to stereotype «dataType». Class *AbstractOpticalProperty* and all its subclasses are mapped to table NG3_OPTICAL_PROPERTY.

7.6.6 Energy ADE 3.0 mapping overview

Table 4 presents the overview of the mapping for all Energy ADE 3.0 modules and classes. Information is provided about the tables (and, if required, the ancillary/associated tables) needed to store each class properties. Finally, for each class, the specific delete and envelope functions are provided. The former can be used to delete an object (or a set thereof) belonging to that class by means of SQL queries, the latter can be used to compute the 3D bounding bpx of a certain feature and to optionally update the envelope field in the CITYOBJECT table in order to speed up bounding-box based spatial queries. More details about delete functions are provided in the next section 7.6.7.
Specific mapping details are provided in section 7.7, in which a colour coding is adopted: in the UML diagrams, class(es) mapped to a table are highlighted using an orange background and writing the name of the corresponding table in red. Associations tables, generally needed to represent many-to-many relations, are depicted using green boxes.

Table 4. General overview of Energy ADE 3.0 classes, their mapping to tables, and the corresponding delete functions.

Package	Class name	Tables	Associated table(s)	Delete function	Envelope function
DataTypes	AbstractOpticalProperty	NG3_OPTICAL_PROPERTY		del_ng3_optical_property(…)	
DataTypes	**Emissivity**	NG3_OPTICAL_PROPERTY		del_ng3_optical_property(…)	
DataTypes	**Transmittance**	NG3_OPTICAL_PROPERTY		del_ng3_optical_property(…)	
DataTypes	**Reflectance**	NG3_OPTICAL_PROPERTY		del_ng3_optical_property(…)	
DataTypes	AbstractQualifiedAttribute	NG3_QUALIFIED_ATTRIBUTE		del_ng3_qualified_attribute(…)	
DataTypes	**QualifiedHeight**	NG3_QUALIFIED_ATTRIBUTE		del_ng3_qualified_attribute(…)	
DataTypes	**QualifiedArea**	NG3_QUALIFIED_ATTRIBUTE		del_ng3_qualified_attribute(…)	
DataTypes	**QualifiedVolume**	NG3_QUALIFIED_ATTRIBUTE		del_ng3_qualified_attribute(…)	
DataTypes	**Suitability**	NG3_SUITABILITY		del_ng3_suitability(…)	
Core	ADE _CityObject	(CITYOBJECT) + NG3_CITYOBJECT		del_ng3_cityobject(…): only ADE data del_cityobject(…): whole object	env_cityobject(…)
Core	ADE _AbstractBuilding	(CITYOBJECT + BUILDING) + NG3_BUILDING	NG3_QUALIFIED_ATTRIBUTE	del_ng3_cityobject(…): only ADE data del_building(…): whole object	env_building(…)
Core	**WeatherData**	CITYOBJECT + NG3_WEATHER_DATA		del_ng3_weather_data(…)	env_ng3_weather_data(…)
Core	AbstractLayeredConstruction	CITYOBJECT + NG3_LAYERED_CONSTRUCTION		del_ng3_layered_construction(…)	
Core	AbstractDevice	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Core	AbstractResource	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Core	**UtilityNetworkConnection**	CITYOBJECT + NG3_UTL_NTW_CONNECTION		del_ng3_utl_ntw_connection(…)	
Core	**CityObjectRelation**	NG3_CTYOBJ_RELATION		del_ng3_cty_obj_relation(…)	
Core	**DeviceOperation**	CITYOBJECT + NG3_DEVICE_OPERATION		del_ng3_device_operation(…)	
Core	AbstractBuildingPartition	CITYOBJECT + NG3_BUILDING_PARTITION	NG3_QUALIFIED_ATTRIBUTE, SURFACE_GEOMETRY	del_ng3_building_partition(…)	env_ng3_building_partition(…)
Core	AbstractBuildingUnit	CITYOBJECT + NG3_BUILDING_PARTITION		del_ng3_building_partition(…)	env_ng3_building_partition(…)
Core	AbstractUsageZone	CITYOBJECT + NG3_BUILDING_PARTITION		del_ng3_building_partition(…)	env_ng3_building_partition(…)
Core	AbstractUsageZone	CITYOBJECT + NG3_BUILDING_PARTITION		del_ng3_building_partition(…)	env_ng3_building_partition(…)
Core	**EnergyPerformanceCertificate**	CITYOBJECT + NG3_ENERGY_PERF_CERT		del_ng3_energy_perf_cert(…)	
Core	**RefurbishmentMeasure**	CITYOBJECT + NG3_REFURBISHMENT_MEASURE		del_ng3_refurbishment_measure(…)	
Core	AbstractLibrary	CITYOBJECT + NG3_LIBRARY		del_ng3_library(…)	env_ng3_library(…)

Package	Class name	Tables	Associated table(s)	Delete function	Envelope function
Resources	**Energy**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**Water**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**Waste**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**Food**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**ConstructionMaterial**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**OtherResource**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Resources	**UrbanSpace**	CITYOBJECT + NG3_RESOURCE		del_ng3_resource(…)	
Building Physics	**ThermalZone**	CITYOBJECT + NG3_BUILDING_PARTITION		del_ng3_building_partition(…)	
Building Physics	ADE _BoundarySurface	(CITYOBJECT + THEMATIC_SURFACE) + NG3_THEMATIC_SURFACE		del_ng3_thematic_surface(…): only ADE data del_thematic_surface(…):whole object	env_thematic_surface(…)
Building Physics	**PartyWallSurface**	CITYOBJECT + THEMATIC_SURFACE + NG3_THEMATIC_SURFACE	NG3_THEM_SURF_TO_THERMAL_ZONE	del_ng3_thematic_surface(…): only ADE data	env_thematic_surface(…)
Building Physics	**IntermediateFloorSurface**	CITYOBJECT + THEMATIC_SURFACE + NG3_THEMATIC_SURFACE	NG3_THEM_SURF_TO_THERMAL_ZONE	del_thematic_surface(…):whole object	env_thematic_surface(…)
Building Physics	**BasementCeilingSurface**	CITYOBJECT + THEMATIC_SURFACE + NG3_THEMATIC_SURFACE	NG3_THEM_SURF_TO_THERMAL_ZONE	del_ng3_thematic_surface(…): only ADE data	env_thematic_surface(…)
Building Physics	**AtticFloorSurface**	CITYOBJECT + THEMATIC_SURFACE + NG3_THEMATIC_SURFACE	NG3_THEM_SURF_TO_THERMAL_ZONE	del_thematic_surface(…):whole object	env_thematic_surface(…)
Building Physics	ADE _Opening	(CITYOBJECT + OPENING) + NG3_OPENING		del_ng3_opening(…): only ADE data del_opening(…): whole object	env_opening(…)
Occupancy	**UsageZone**	CITYOBJECT + NG3_BUILDING_PARTITION		del_ng3_building_partition(…)	env_ng3_building_partition(…)
Occupancy	**BuildingUnit**	CITYOBJECT + NG3_BUILDING_PARTITION	NG3_ADDRESS_TO_BUILDING_UNIT	del_ng3_building_partition(…)	env_ng3_building_partition(…)
Occupancy	**Occupants**	CITYOBJECT + NG3_OCCUPANTS		del_ng3_occupants(…)	
Devices	**HeatPump**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**Boiler**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**LightingDevice**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**GenericDevice**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**GenericElectricalDevice**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**MovableShadingDevice**	CITYOBJECT + NG3_DEVICE		del_ng3_device(…)	env_ng3_device(…)
Devices	**ThermalStorageDevice**	CITYOBJECT + NG3_DEVICE + NG3_STORAGE_DEVICE		del_ng3_storage_device(…)	env_ng3_device(…)
Devices	**ElectricalStorageDevice**	CITYOBJECT + NG3_DEVICE + NG3_STORAGE_DEVICE		del_ng3_storage_device(…)	env_ng3_device(…)

Package	Class name	Tables	Associated table(s)	Delete function	Envelope function
Devices	AbstractSolarCollector	CITYOBJECT + NG3_DEVICE + NG3_SOLAR_COLLECTOR	SURFACE_GEOMETRY	del_ng3_solar_collector(…)	env_ng3_device(…)
Devices	**GenericSolarCollector**	CITYOBJECT + NG3_DEVICE + NG3_SOLAR_COLLECTOR		del_ng3_solar_collector(…)	env_ng3_device(…)
Devices	**SolarThermalCollector**	CITYOBJECT + NG3_DEVICE + NG3_SOLAR_COLLECTOR		del_ng3_solar_collector(…)	env_ng3_device(…)
Devices	**PhotovoltaicCollector**	CITYOBJECT + NG3_DEVICE + NG3_SOLAR_COLLECTOR		del_ng3_solar_collector(…)	env_ng3_device(…)
Devices	**PhotovoltaicThermalCollector**	CITYOBJECT + NG3_DEVICE + NG3_SOLAR_COLLECTOR		del_ng3_solar_collector(…)	env_ng3_device(…)
Urban Function Area	**UrbanFunctionArea**	CITYOBJECT + CITYOBJECTGROUP + NG3_URBAN_FUNCTION_AREA		del_ng3_urban_function_area(…)	env_cityobjectgroup(…)
Weather Station	**WeatherStation**	CITYOBJECT		del_ng3_cityobject(…)	env_cityobject(…)
Layered Construction	**LayeredConstruction**	CITYOBJECT + NG3_LAYERED_CONSTRUCTION	NG3_OPTICAL_PROPERTY	del_ng3_layered_construction(…)	
Layered Construction	**ReverseLayeredConstruction**	CITYOBJECT + NG3_LAYERED_CONSTRUCTION		del_ng3_layered_construction(…)	
Layered Construction	**Layer**	CITYOBJECT + NG3_LAYER		del_ng3_layer(…)	
Layered Construction	AbstractMaterial	CITYOBJECT + NG3_MATERIAL		del_ng3_material(…)	
Layered Construction	**Gas**	CITYOBJECT + NG3_MATERIAL		del_ng3_material(…)	
Layered Construction	**SolidMaterial**	CITYOBJECT + NG3_MATERIAL		del_ng3_material(…)	
Layered Construction	**MaterialLibrary**	CITYOBJECT + NG3_LIBRARY		del_ng3_library(…)	env_ng3_library(…)
Layered Construction	**LayeredConstructionLibrary**	CITYOBJECT + NG3_LIBRARY		del_ng3_library(…)	env_ng3_library(…)
Schedule	AbstractSchedule	CITYOBJECT + NG3_SCHEDULE		del_ng3_schedule(…)	
Schedule	**AtomicSchedule**	CITYOBJECT + NG3_SCHEDULE		del_ng3_schedule(…)	
Schedule	**DualSchedule**	CITYOBJECT + NG3_SCHEDULE		del_ng3_schedule(…)	
Schedule	**CompositeSchedule**	CITYOBJECT + NG3_SCHEDULE		del_ng3_schedule(…)	
Schedule	**ScheduleComponent**	CITYOBJECT + NG3_SCHEDULE_COMPONENT		del_ng3_schedule_component(…)	
Schedule	**ScheduleLibrary**	CITYOBJECT + NG3_LIBRARY		del_ng3_library(…)	env_ng3_library(…)
Time Series	AbstractTimeSeries	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	
Time Series	**RegularTimeSeries**	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	

Package	Class name	Tables	Associated table(s)	Delete function	Envelope function
Time Series	**RegularTimeSeriesFile**	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	
Time Series	**DefaultValuesTimeSeries**	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	
Time Series	**DefaultValuesTimeSeriesFile**	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	
Time Series	**SensorConnection**	CITYOBJECT + NG3_TIME_SERIES		del_ng3_time_series(…)	

7.6.7 Energy ADE 3.0 delete functions

In order to tackle the complexity of deleting database objects that may be written in many linked tables, 3DCityDB is shipped with a series of stored procedures that simplify the process[30]. Each 3DCityDB schema comes with its own set of functions. In the case of PostgreSQL/PostGIS, all delete functions are written in PL/pgSQL and are prefixed with "del_". For each class, there are generally two homonymous functions that differ only in terms of input signature: a single object id, or an arrays of ids. In either case the function will take care of deleting the selected object(s) and all its dependent data automatically.
Just like in the "standard" 3DCityDB, delete functions are generated and added to a database schema upon installation of an ADE. The underlying logic and the purpose is the same, the only difference is that all ADE functions contain the database prefix in their names. Table 4 presents, for each class, the specific delete function. Please note that, for all classes having data stored in/starting from table CITYOBJECT, the generic delete function **del_cityobject(…)** can be used too. As shown in Listing 3, objects can be deleted in different ways, therefore only some examples are provided.

```
-- Delete a qualified attribute using the id
SELECT citydb.del_ng3_optical_property(id) FROM citydb.ng3_optical_property WHERE id=33;
-- Delete all grossFloorArea attributes belonging to buildings
SELECT citydb.del_ng3_optical_property(id) FROM citydb.ng3_optical_property WHERE type='grossFloorArea' AND building_id IS NOT
NULL;

-- Delete an EnergyPerformanceCertificate object using the id
SELECT citydb.del_ng3_energy_perf_cert(id) FROM citydb.ng3_energy_perf_cert WHERE id=47;
SELECT citydb.del_ng3_energy_perf_cert(id) FROM citydb.cityobject WHERE id=47;
-- Delete an EnergyPerformanceCertificate object using the generic delete function del_cityobject(…)
SELECT citydb.del_cityobject(id) FROM citydb.cityobject WHERE id=47;

-- Delete a MaterialLibrary object (and all children objects) using the id
SELECT citydb.del_ng3_library(id) FROM citydb.ng3_library WHERE id=3;
SELECT citydb.del_ng3_library(id) FROM citydb.cityobject WHERE id=3;
-- Delete a MaterialLibrary object (and all children objects) using the gmlid
SELECT citydb.del_ng3_library(id) FROM citydb.cityobject WHERE gmlid='id_material_library_1';
-- Delete a MaterialLibrary object using the generic delete function del_cityobject(…)
SELECT citydb.del_cityobject(id) FROM citydb.cityobject WHERE id=3;
SELECT citydb.del_cityobject(id) FROM citydb.cityobject WHERE gmlid='id_material_library_1';

-- Delete a UrbanFunctionArea object using the id
SELECT citydb.del_ng3_urban_function_area(id) FROM citydb.ng3_urban_function_area WHERE id=112;
SELECT citydb.del_ng3_urban_function_area(id) FROM citydb.cityobject WHERE id=112;
SELECT citydb.del_cityobject(id) FROM citydb.cityobject WHERE id=112;
-- Optionally, clean up global appearances
SELECT citydb.cleanup_appearances();

-- Delete a ThermalZone object object (and all children objects) using the id
SELECT citydb.del_ng3_building_partition(id) FROM citydb.ng3_building_partition WHERE id=98;
SELECT citydb.del_ng3_building_partition(id) FROM citydb.cityobject WHERE id=98;
SELECT citydb.del_cityobject(id) FROM citydb.cityobject WHERE id=98;
-- Optionally, clean up global appearances
SELECT citydb.cleanup_appearances();

-- Delete all ADE attributes of a thematic surface using the id
SELECT citydb.del_ng3_thematic_surface(id) FROM citydb.ng3_thematic_surface WHERE id=233;
```

Listing 3. Example of SQL queries to delete objects stored in the 3DCityDB.

[30] https://3dcitydb-docs.readthedocs.io/en/latest/3dcitydb/sproc/citydb-delete.html

7.6.8 Energy ADE 3.0 envelope functions

In order to compute the maximum 3D bounding box of any of the objects stored in table CITYOBJECT (i.e. all classes derived from *_CityObject* and the ADE classes derived from *_Feature*), the 3DCityDB is shipped with a series of stored procedures that simplify the process[31]. Each 3DCityDB schema comes with its own set of functions. In the case of PostgreSQL/PostGIS, all envelope functions are written in PL/pgSQL and are prefixed with "env_". For each class, there is a function that requires as input the id of the object. Additionally, the envelope column of table CITYOBJECT can be written/updated in order to allow for faster, bounding-box-based spatial queries.
Just like in the "standard" 3DCityDB, envelope functions are generated and added to a database schema upon installation of an ADE. The underlying logic and the purpose is the same, the only difference is that all ADE functions contain the database prefix in their names. Table 4 presents the envelope functions associated to each city object or feature object. Please note that, for all classes having data stored in/starting from table CITYOBJECT, the generic envelope function **env_cityobject(…)** can be used too. As shown in Listing 4, the envelope can be computed in different ways, therefore only some examples are provided.

```
-- Compute the 3D bounding box of building openings
SELECT citydb.env_opening(id) FROM citydb.opening;
-- Compute the 3D bounding box of building openings and update the envelope column of table CITYOBJECT
-- Please note the additional input parameter
SELECT citydb.env_opening(id, 1) FROM citydb.opening;
-- Same as before, but using the generic envelope function
SELECT citydb.env_cityobject(id, 1) FROM citydb.opening;

-- Compute the 3D bounding box of building thematic surfaces
SELECT citydb.env_thematic_surface(id) FROM citydb.thematic_surface;
-- Compute the 3D bounding box of building thematic surfaces and update table CITYOBJECT
-- Will take care of the children building openings too
SELECT citydb.env_cityobject(id, 1) FROM citydb.thematic_surface;

-- Compute the 3D bounding box of UsageZone objects
WITH o AS (SELECT id FROM citydb.objectclass WHERE classname='UsageZone')
SELECT citydb.env_ng3_building_partition(bp.id, 1) FROM citydb.ng3_building_partition AS bp JOIN o ON (bp.objectclass_id = o.id);
-- Compute the 3D bounding box of ThermalZone objects and update table CITYOBJECT
-- Will take care of the children building thematic surfaces, openings too
WITH o AS (SELECT id FROM citydb.objectclass WHERE classname='ThermalZone')
SELECT citydb.env_ng3_building_partition(bp.id, 1) FROM citydb.ng3_building_partition AS bp JOIN o ON (bp.objectclass_id = o.id);
-- Compute the 3D bounding box of all building partition objects
SELECT citydb.env_ng3_building_partition(id) FROM citydb.ng3_building_partition;
-- Compute the 3D bounding box of all building partition objects and update table CITYOBJECT
SELECT citydb.env_cityobject(id) FROM citydb.ng3_building_partition;

-- Compute the 3D bounding box of Device objects
SELECT citydb.env_ng3_device(id, 1) FROM citydb.ng3_device;

-- Compute the 3D bounding box of Building(Part) objects
-- Will also take care of all children
SELECT citydb.env_building(id, 1) FROM citydb.building;

-- Compute the 3D bounding box of UrbanFunctionArea objects
SELECT citydb.env_cityobjectgroup(id, 1) FROM citydb.ng3_urban_function_area;
```

Listing 4. Example of SQL queries to compute the 3d bounding box of different objects in the 3DCityDB.

[31] https://3dcitydb-docs.readthedocs.io/en/latest/3dcitydb/sproc/citydb-envelope.html

7.7 From OO- to ER-model: Mapping details

This section provides further details on the mapping between the Object-Oriented model, depicted using UML diagrams, and the database implementation of the Energy ADE 3.0, depicted using typical Entity-Relationship diagrams. For each module, the mapping is also presented graphically in the UML diagrams. **Orange "boxes"** identify the class(es) mapped to the same table, and the table name is provided too. **Light green "boxes"** are used to identify many-to-many relations that are mapped to an association table. Finally, some examples are provided with data stored in the 3DCityDB. The data shown comes from the Alderaan dataset described before. Data has been imported into the 3DCityDB (extended with the Energy ADE 3.0 tables) using the FME workbench described in section 8.

BEWARE: It is important to remark that the dataset used for the examples presented in the following section have been created only for testing purposes. Therefore, some parameters may have no real physical meaning, or could simply be unrealistic. Nevertheless, the purpose is to provide examples of objects of nearly every class and every attribute of the Energy ADE 3.0.

7.7.1 Time series module

Following the mapping rules described in section 7.6.4, and as depicted in Figure 45, all classes of the Time series module are grouped together and mapped to one single target table: NG3_TIME_SERIES. All attributes up to class *_Feature* are written to table CITYOBJECT. Then all attributes from class *AbstractTimeSeries* are written to table NG3_TIME_SERIES. This means that, depending on the specific class, some columns will be written, while others may remain empty. Figure 46 depicts the resulting ER-model.

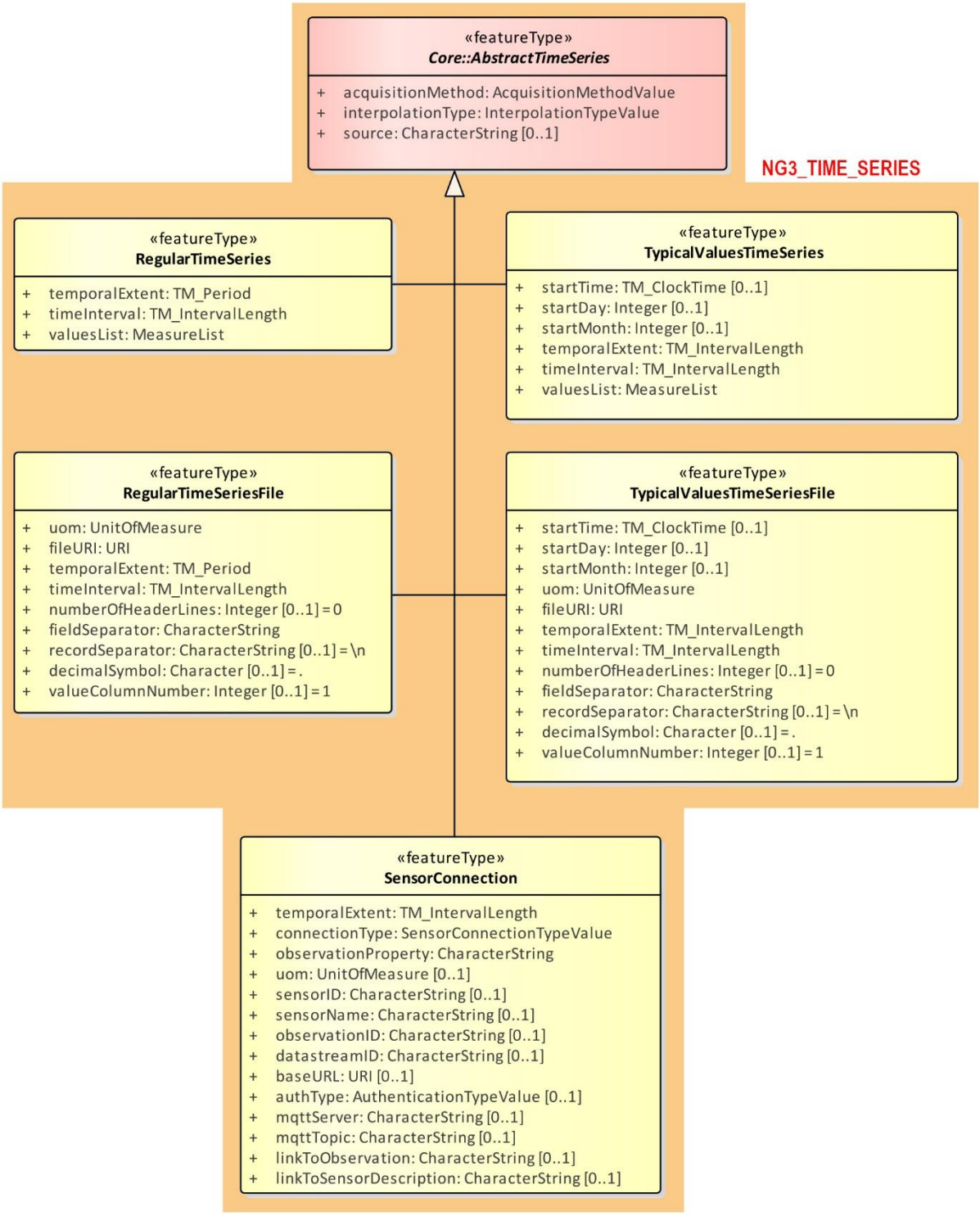


Figure 45. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Time series module.

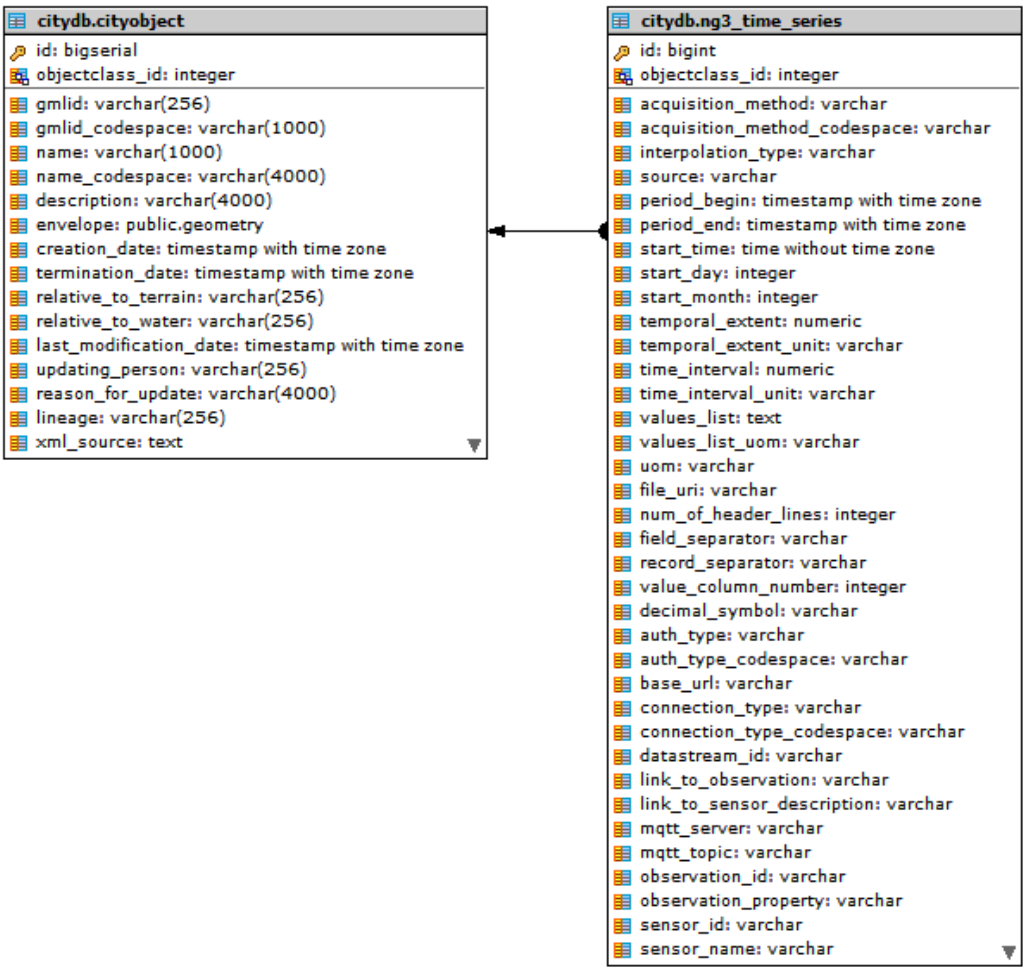


Figure 46. ER-model of the Energy ADE 3.0 Time series module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 5, some time series data are extracted from the 3DCityDB and shown in Figure 47. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of time series data, one for each class of Time series module
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname IN ('RegularTimeSeries', 'RegularTimeSeriesFile', 'TypicalValuesTimeSeries', 'TypicalValuesTimeSeriesFile', 'SensorConnection')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, ts.* FROM citydb.ng3_time_series AS ts JOIN co ON (ts.id = co.id)
ORDER BY co.classname, co.id;
```

Listing 5. Example of SQL query to extract time series data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying ('	name character varying (1000)	name_codespace character varying (	description character varying (4000)	id bigint	objectclass_id integer
765	11021	RegularTimeSeries	id_regular_time_series_1	[null]	RegularTimeSeries 1	[null]	This is RegularTimeSeries 1	765	11021
767	11022	RegularTimeSeriesFile	id_regular_time_series_file_1	[null]	RegularTimeSeriesFile 1	[null]	This is a RegularTimeSeriesFile 1	767	11022
769	11025	SensorConnection	id_sensor_connection_1	[null]	SensorConnection 1	[null]	This is a SensorConnection 1	769	11025
758	11023	TypicalValuesTimeSeries	id_default_values_time_series_6	[null]	DefaultValuesTimeSeries 6	[null]	This is DefaultValuesTimeSeries 6	758	11023
773	11024	TypicalValuesTimeSeriesFile	id_typical_values_time_series_file_2	[null]	TypicalValuesTimeSeriesFile 2	[null]	This is a TypicalValuesTimeSeriesFile 2	773	11024

acquisition_method character varying	acquisition_method_codespace character varying	interpolation_type character varying	source character varying	period_begin timestamp with time zone	period_end timestamp with time zone	start_time time without time zone	start_day integer	start_month integer	temporal_extent numeric
simulation	codespace_2	averageInSucceedingInterval	source_1	2011-01-01 00:00:00+01	2012-01-01 00:00:00+01	[null]	[null]	[null]	[null]
simulation	simulation_codespace	averageInSucceedingInterval	source_1	2011-01-01 00:00:00+01	2012-01-01 00:00:00+01	[null]	[null]	[null]	[null]
measurement	measurement_codespace	averageInSucceedingInterval	source_1	[null]	[null]	[null]	[null]	[null]	1
calibratedSimulation	codespace_22	discontinuous	source_22	[null]	[null]	[null]	[null]	[null]	1
simulation	simulation_codespace	averageInSucceedingInterval	source_2	[null]	[null]	00:00:00	1	1	1

temporal_extent_unit character varying	time_interval numeric	time_interval_unit character varying	values_list text	values_list_uom character varying	uom character varying	file_uri character varying	num_of_header_lines integer	field_separator character varyin	record_separator character varying	value_column_number integer
[null]	0.083	year	5 6 7 10 15 18 19 20 18 ...	Degrees Celsius	[null]	[null]	[null]	[null]	[null]	[null]
[null]	1	day	[null]	[null]	kWh/(m2*day)	//giorgio/results_irradiation.csv	10	,	;	5
year	[null]	[null]	[null]	[null]	units	[null]	[null]	[null]	[null]	[null]
day	1	hour	1 2 3 4 5 6 7 8 9 10 11 1...	unit	[null]	[null]	[null]	[null]	[null]	[null]
year	1	day	[null]	[null]	Scale	//giorgio/results_cloudiness.c...	3	,	;	2

decimal_symbol character varying	auth_type character varying	auth_type_codespace character varying	base_url character varying	connection_type character varying	connection_type_codespace character varying	datastream_id character varying	link_to_observation character varying	link_to_sensor_description character varying	mqtt_server character varying	mqtt_topic character varying
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
.	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
[null]	auth type	auth_type_codespa...	base url	connection type	conn-type_codespace	datastream_id	ling to observation	link to sensor description	mqtt server	mqtt topic
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
.	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]

observation_id character varying	observation_property character varying	sensor_id character varying	sensor_name character varying
[null]	[null]	[null]	[null]
[null]	[null]	[null]	[null]
observation_id	observation proper...	sensor_id	sensor name
[null]	[null]	[null]	[null]
[null]	[null]	[null]	[null]

Figure 47. Example of time series data in the 3DCityDB.

7.7.2 Schedules module

In the Schedule module, different mapping rules are applied. Following the rules in section 7.6.3, class *ScheduleLibrary* is mapped to target table NG3_LIBRARY. All attributes up to class *_CityObject* are written to table CITYOBJECT, then all attributes from class *AbstractLibrary* are written to table NG3_LIBRARY.

When it comes to the classes derived from class *AbstractSchedule*, the mapping rules of section 7.6.4 apply. All classes are grouped together and mapped to one single target table: NG3_SCHEDULE. All attributes up to class *_Feature* are written to table CITYOBJECT. Then, all attributes from class *AbstractSchedule* are written to table NG3_SCHEDULE. This means that, depending on the specific class, some columns will be written, while others may remain empty.

The relation to the *ScheduleLibrary* is provided by the foreign key in column *library_id*. In the case of the *AtomicSchedule*, the optional relation to the *AbstractTimeSeries* data is provided by the foreign key in column *time_series_id*. Alternatively, column *constant_value* and *constant_value_uom* must be used to store the *constantValue*.

The same mapping rules of *AbstractSchedule* apply also to class *ScheduleComponent*: all attributes up to class *_Feature* are written to table CITYOBJECT. Then, all attributes of class *ScheduleComponent* are written to table NG3_SCHEDULE_COMPONENT. The relation to the parent *CompositeSchedule* is provided by the foreign key in column *parent_schedule_id*. The relation to the child *AbstractSchedule* is provided by the foreign key in column *schedule_id*.

Figure 48 represents graphically the OO-to-ER-model mapping of the Schedule module. Figure 49 depicts the resulting ER-model.

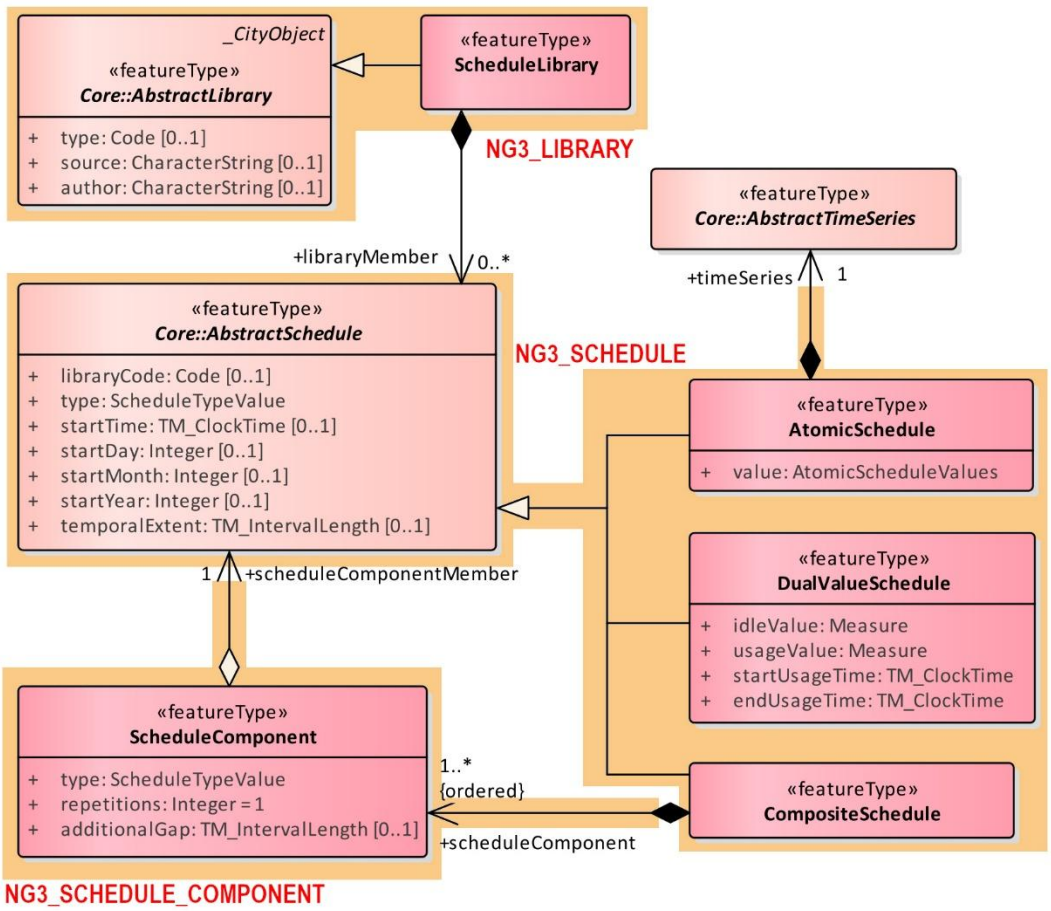


Figure 48. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Schedule module.

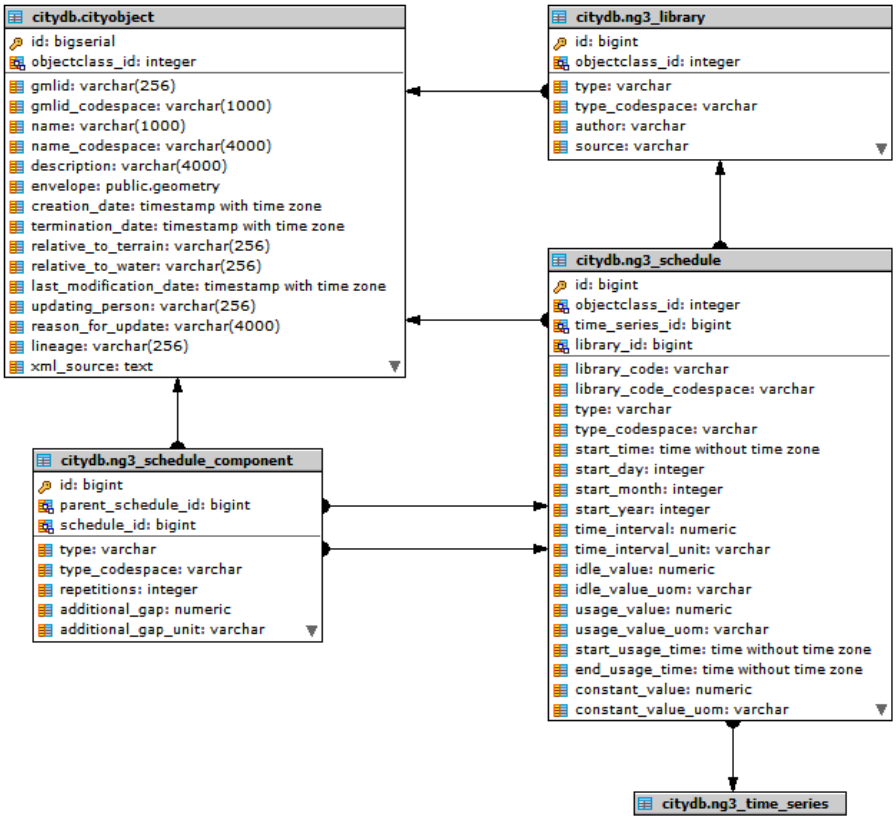


Figure 49. ER-model of the Energy ADE 3.0 Schedule module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 6, some schedule data are extracted from the 3DCityDB and shown in Figure 50. The resulting table is split over multiple lines due to its width.

```
-- Extract the library containing the schedules
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname = 'ScheduleLibrary'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_library AS t JOIN co ON (t.id = co.id)
ORDER BY co.id;
```

Listing 6. Example of SQL query to extract the *ScheduleLibrary* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying ('	name character varying (10	name_codespace character varying (
652	11143	ScheduleLibrary	id_schedule_library_1	[null]	Schedule Library 1	[null]

description character varying (4000)	id bigint	objectclass_id integer	type character varying	type_codespace character varying	author character varying
This is Schedule Library 1	652	11143	schedule_library_type	schedule_library_type_codeSpace	Giorgio Agugiaro

source character varying
Coruscant Schedule Library

Figure 50. Example of *ScheduleLibrary* data in the 3DCityDB.

Similarly, using the SQL query in Listing 7, some schedule data are extracted from the 3DCityDB and shown in Figure 51. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of schedule data, one for each class of Schedules module
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname IN ('DualValueSchedule', 'AtomicSchedule', 'CompositeSchedule')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_schedule AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname, co.id;
```

Listing 7. Example of SQL query to extract schedule data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying (4000)
413	11031	AtomicSchedule	id_atomic_schedule_1	[null]	AtomicSchedule 1	[null]
425	11033	CompositeSchedule	id_composite_schedule...	[null]	CompositeSchedule 2	[null]
434	11032	DualValueSchedule	id_dual_value_schedule_8	[null]	DualValueSchedule 8	[null]

description character varying (4000)	id bigint	objectclass_id integer	library_code character varying	library_code_codespace character varying
This is AtomicSchedule 1 for a year, connected to a timeseries of 1 yearly value	413	11031	atom_sched_code_1	schedule_library_codeSpace
This is CompositeSchedule 2, for 1 year, composed of 4 seasons	425	11033	comp_sched_code_2	schedule_library_codeSpace
This is DualValueSchedule 8 for a year	434	11032	dual_value_sched_code_8	schedule_library_codeSpace

type characte	type_codespace character varying	start_time time without time zone	start_day integer	start_month integer	start_year integer	time_interval numeric	time_interval_unit character varying	constant_value numeric
year	schedule_type_codeSpace	00:00:00	1	1	[null]	1	year	[null]
week	schedule_type_codeSpace	00:00:00	[null]	[null]	[null]	7	day	[null]
year	schedule_type_codeSpace	00:00:00	1	1	[null]	1	year	[null]

constant_value_uom character varying	idle_value numeric	idle_value_uom character varying	usage_value numeric	usage_value_uom character varying	start_usage_time time without time zone	end_usage_time time without time zone	time_series_id bigint	library_id bigint
[null]	[null]	[null]	[null]	[null]	[null]	[null]	751	652
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	652
[null]	0.5	unit interval	1	unit interval	09:00:00	18:00:00	[null]	652

Figure 51. Example of schedule data in the 3DCityDB.

Finally, using the SQL query in Listing 8, some schedule component data are extracted from the 3DCityDB and shown in Figure 52. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of ScheduleComponent data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname = 'ScheduleComponent' LIMIT 2)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_schedule_component AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname, co.id;
```

Listing 8. Example of SQL query to extract *ScheduleComponent* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (1	name character varying (1000)
645	11034	ScheduleComponent	id_schedule_component_1	[null]	ScheduleComponent 1 containing a repetition of 5 times a daily sched...
646	11034	ScheduleComponent	id_schedule_component_2	[null]	ScheduleComponent 2 containing a repetition of 2 times a daily sched...

name_codespace character varying (4	description character varying (4000)	id bigint	type character varyir	type_codespace character varying	repetitions integer	additional_gap numeric	additional_gap_unit character varying	parent_schedule_id bigint	schedule_id bigint
[null]	This is ScheduleComponent 1	645	weekDay	schedule_type_codeSpace	5	0	day	424	418
[null]	This is ScheduleComponent 2	646	weekendDay	schedule_type_codeSpace	2	0	day	424	419

Figure 52. Example of *ScheduleComponent* data in the 3DCityDB.

7.7.3 Layered construction module

In the Layered construction module, different mapping rules are applied. Following the rules in section 7.6.3, classes *MaterialLibrary* and *LayeredConstructionLibrary* are mapped to target table NG3_LIBRARY. All attributes up to class *_CityObject* are written to table CITYOBJECT, then all attributes from class *AbstractLibrary* are written to table NG3_LIBRARY.

When it comes to the classes derived from class *AbstractMaterial*, the mapping rules of section 7.6.4 apply. All classes are grouped together and mapped to one single target table: NG3_MATERIAL. All attributes up to class *_Feature* are written to table CITYOBJECT. Then, all attributes from class *AbstractMaterial* are written in table NG3_MATERIAL. This means that, depending on the specific class, some columns will be written, while others may remain empty. The relation to the *MaterialLibrary* is provided by the foreign key in column *library_id*.

When it comes to the classes derived from class *AbstractLayeredConstruction*, the mapping rules of section 7.6.4 apply. All classes are grouped together and mapped to one single target table: NG3_LAYERED_CONSTRUCTION. All attributes up to class *_Feature* are written in table CITYOBJECT. Then, all attributes from class *AbstractLayeredConstruction* are written in table NG3_LAYERED_CONSTRUCTION. This means that, depending on the specific class, some columns will be written, while others may remain empty. The relation to the *LayeredConstructionLibrary* is provided by the foreign key in column *library_id*. The relation from *ReverseLayeredConstruction* to *LayeredConstruction* is provided by the foreign key in column *layered_construction_id*. *Emissivity*, *Reflectance* and *Transmittance* properties are written in table NG3_OPTICAL_PROPERTY and referenced by means of foreign key *layered_construction_id*.

The same mapping rules of *AbstractLayeredConstruciton* apply also to class *Layer*: all attributes up to class *_Feature* are written to table CITYOBJECT. Then, all attributes of class *Layer* are written to table NG3_LAYER. The relation to the parent *LayeredConstruciton* is provided by the foreign key in column *layered_construction_id*. The relation to the child *AbstractMaterial* is provided by the foreign key in column *material_id*.

Figure 53 represents graphically the OO-to-ER-model mapping of the Schedule module. Figure 54 and Figure 55 depict the resulting ER-models.

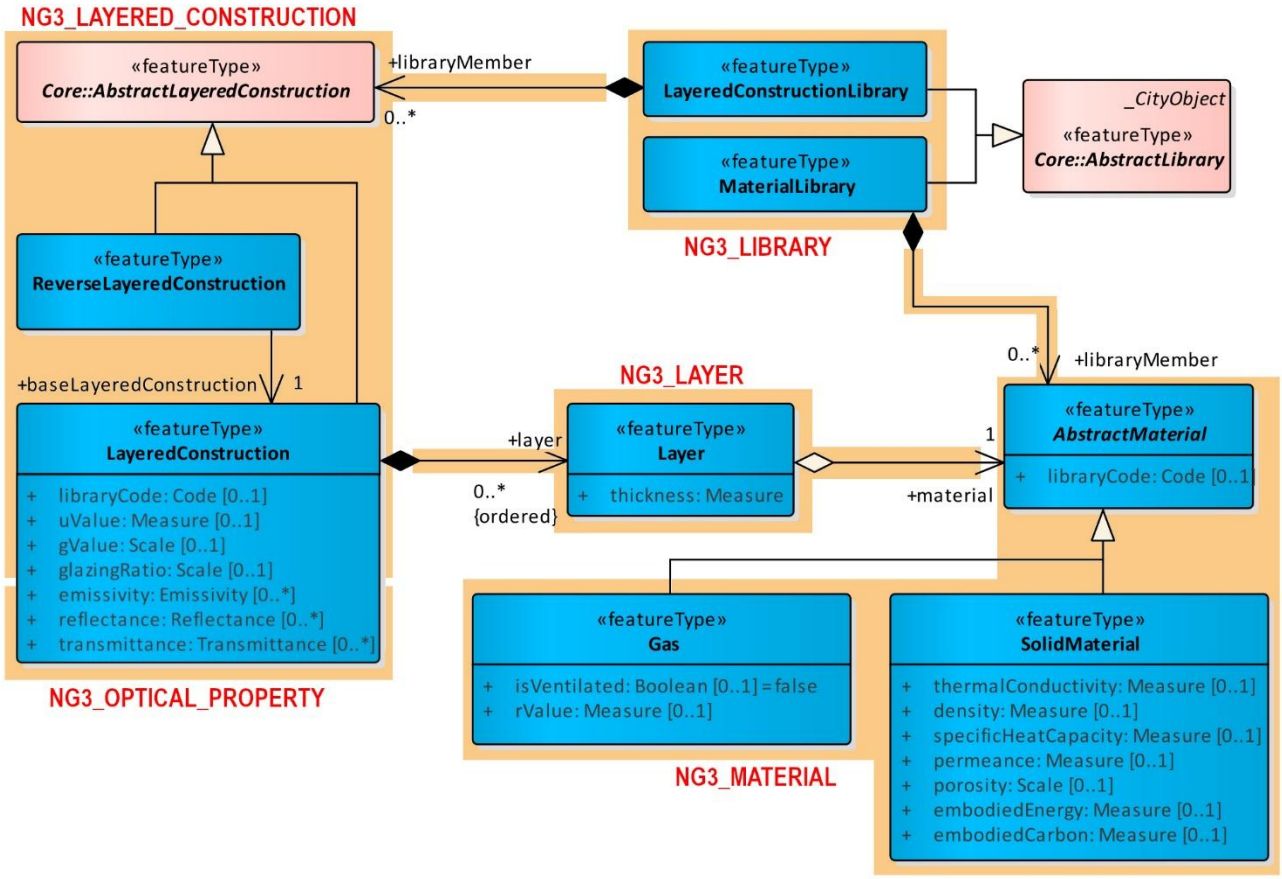


Figure 53. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Layered construction module.

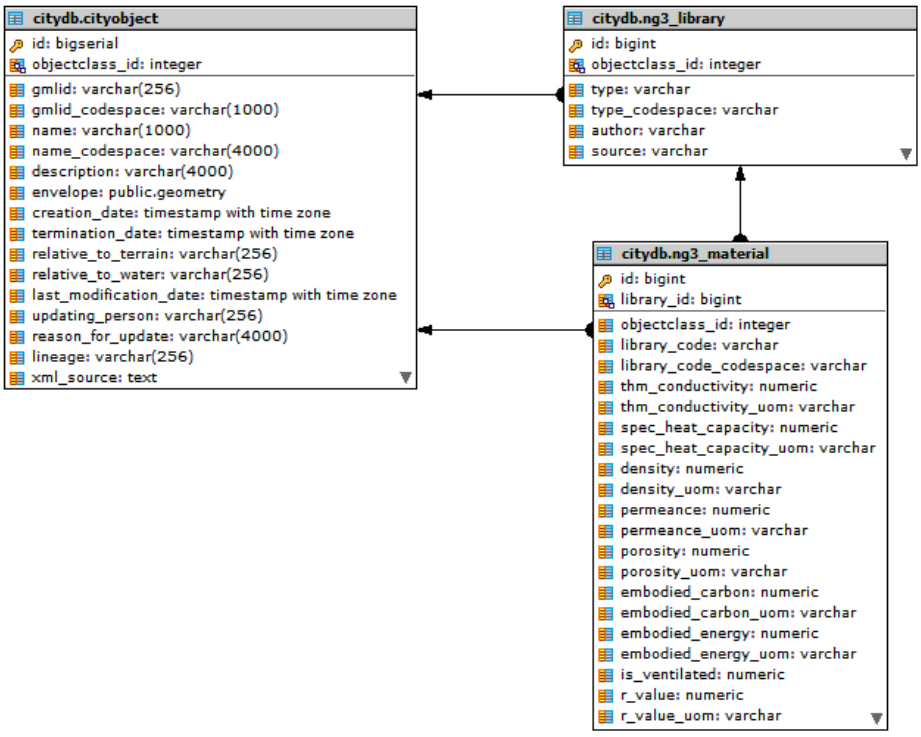


Figure 54. ER-model of the Energy ADE 3.0 Layered construction module, focusing on the tables related to classes *MaterialLibrary* and *AbstractMaterial*.

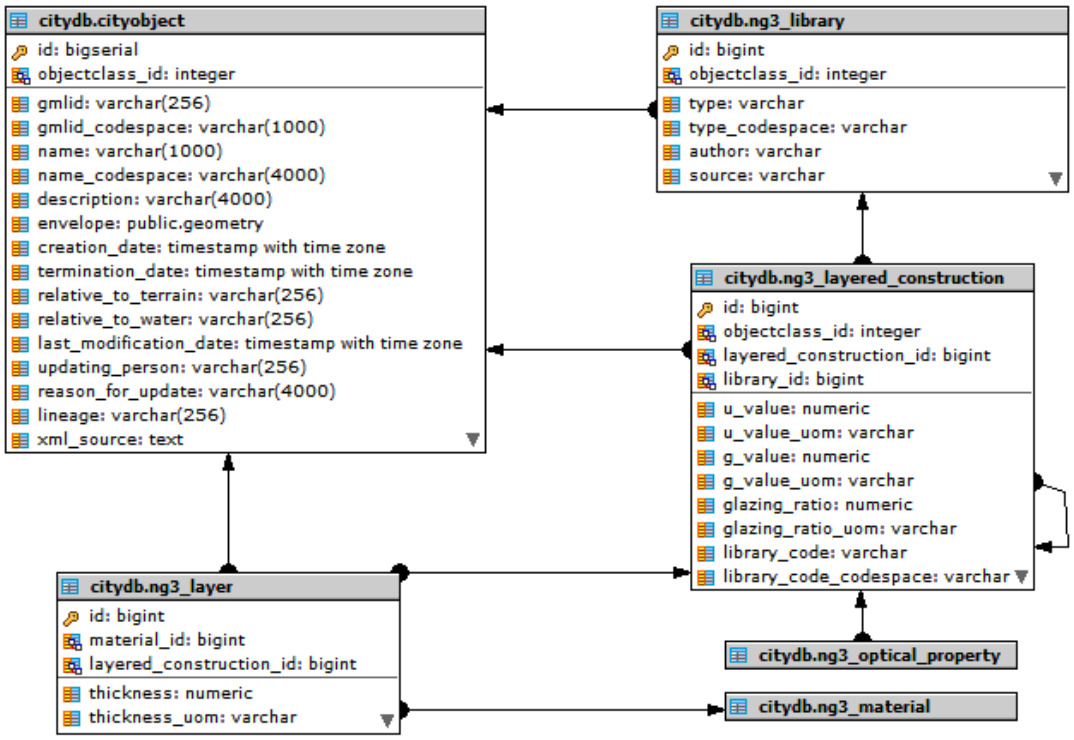


Figure 55. ER-model of the Energy ADE 3.0 Layered Construction module, focusing on the tables related to classes LayeredConstructionLibrary and *AbstractLayeredConstruction*.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 9, some data for *MaterialLibrary* and *LayeredConstructionLibrary* are extracted from the 3DCityDB and shown in Figure 56. The resulting table is split over multiple lines due to its width.

```
-- Extract the libraries for materials and layered constructions
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname IN ('MaterialLibrary', 'LayeredConstructionLibrary')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_library AS t JOIN co ON (t.id = co.id)
ORDER BY co.id;
```

Listing 9. Example of SQL query to extract the *MaterialLibrary* and *LayeredConstructionLibrary* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
478	11141	LayeredConstructionLibrary	id_layered_construction_library_1	[null]	Layered Construction Library 1	[null]
479	11142	MaterialLibrary	id_material_library_1	[null]	Material Library 1	[null]

description character varying (4000)	id bigint	objectclass_id integer	type character varying	type_codespace character varying	author character varying
This is Layered Construction Library 1	478	11141	layered_construction_library_type	layered_construction_library_type_codeSpace	Giorgio Agugiaro
This is Material Library 1	479	11142	material_library_type	material_library_type_codeSpace	Giorgio Agugiaro

source character v
TABULA
TABULA

Figure 56. Example of *MaterialLibrary* and *LayeredConstructionLibrary* data in the 3DCityDB.

Using the SQL query in Listing 10, some material data are extracted from the 3DCityDB and shown inFigure 57. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of materials data, one for each class
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname IN ('Gas', 'SolidMaterial')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_material AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname, co.id;
```

Listing 10. Example of SQL query to extract material data from the database.

co_id bigint	objectclass_id integer	classname character varyi	gmlid character varying (256)	gmlid_codespace character varying	name character varying (1(	name_codespace character varying (4	description character varying (4000)	id bigint
454	11056	Gas	id_gas_2	[null]	Gas 2	[null]	This is Gas 2	454
732	11055	SolidMaterial	id_solid_material_18	[null]	SolidMaterial 18	[null]	This is SolidMaterial 18	732

objectclass_id integer	library_code character varying	library_code_codespace character varying	thm_conductivity numeric	thm_conductivity_uom character varying	spec_heat_capacity numeric	spec_heat_capacity_uom character varying	density numeric
11056	gas_code_2	gascode_codeSpace	[null]	[null]	[null]	[null]	[null]
11055	solid_material_code_18	gascode_codeSpace	0.7	W/(K*m)	0.9	J/(kg*K)	1400

density_uom character varyi	permeance numeric	permeance_uom character varying	porosity numeric	porosity_uom character varyi	embodied_carbon numeric	embodied_carbon_uom character varying	embodied_energy numeric	embodied_energy_uom character varying
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
kg/m^3	0.9	xxx	0.05	ratio	0.5	kgCO2/kg	9.4	kWh/kg

is_ventilated numeric	r_value numeric	r_value_uom character var	library_id bigint
0	8.314	J/K/mol	479
[null]	[null]	[null]	479

Figure 57. Example of material data written in the 3DCityDB.

Similarly, using the SQL query in Listing 11, some *LayeredConstruction* and *ReverseLayeredConstruction* data are extracted from the 3DCityDB and shown in Figure 58. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of LayeredConstruction and ReverseLayeredConstruction data, one for each class
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) WHERE o.classname IN ('LayeredConstruction', 'ReverseLayeredConstruction')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM citydb.ng3_layered_construction AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname, co.id;
```

Listing 11. Example of SQL query to extract *LayeredConstruction* and *ReverseLayeredConstruction* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
473	11051	LayeredConstruction	id_layered_construction_glazing_5	[null]	LayeredConstruction Glazing 5 (no layers)	[null]
643	11052	ReverseLayeredConstruction	id_reverse_layered_construction_ground_1	[null]	ReverseLayeredConstruction Ground 1	[null]

description character varying (4000)	id bigint	objectclass_id integer	u_value numeric	u_value_uom character varyi
This is LayeredConstruction Glazing 5 (without layers as children objects)	474	11051	1.9	W/(K*m^2)
ReverseLayeredConstruction Ground 1 (from inside to outside) (reverse the order of the layers of the linked LayeredConstruction)	644	11052	[null]	[null]

g_value numeric	g_value_uom character vary	glazing_ratio numeric	glazing_ratio_uom character varying	library_code character varying	library_code_codespace character varying	layered_construction_id bigint	library_id bigint
0.7	unit interval	0.95	unit interval	layered_constr_library_code_5	layered_constr_library_codeSpace	[null]	479
[null]	[null]	[null]	[null]	[null]	[null]	475	479

Figure 58. Example of *LayeredConstruction* and *ReverseLayeredConstruction* data in the 3DCityDB.

Finally, using the SQL query in Listing 12, some schedule component data are extracted from the 3DCityDB and shown in Figure 59. The resulting table is split over multiple lines due to its width.

```
-- Extract examples of layer data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'Layer' LIMIT 3
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
t.* FROM citydb.ng3_layer AS t JOIN co ON (t.id = co.id)
ORDER BY co.id;
```

Listing 12. Example of SQL query to extract some *Layer* data from the database.

co_id bigint	objectclass_id integer	classname character va	gmlid character va	gmlid_codespace character varying (1	name character	name_codespace character varying (4	description character varying	id bigint	thickness numeric	thickness_uom character varyin
455	11053	Layer	id_layer_1	[null]	Layer 1	[null]	This is Layer 1	455	40	mm
456	11053	Layer	id_layer_2	[null]	Layer 2	[null]	This is Layer 2	456	45	mm
457	11053	Layer	id_layer_3	[null]	Layer 3	[null]	This is Layer 3	457	80	mm

material_id bigint	layered_construction_id bigint
717	474
718	474
719	474

Figure 59. Example of *Layer* data in the 3DCityDB.

7.7.4 DataTypes module

As explained in section 7.6.5, classes belonging to stereotype «dataType» are mapped to independent tables, each table having its own sequence, primary key, etc.

In the case of the Energy ADE 3.0, all classes derived from class *AbstractOpticalProperty* are mapped to table NG3_OPTICAL_PROPERTY. Column *layered_contruction_id* contains a foreign key to table NG3_LAYERED_CONSTRUCTION.

All classes derived from class *AbstractQualifiedAttribute* are mapped to table NG3_QUALIFIED_ATTRIBUTE. Columns *building_id* and *building_partition_id* contain foreign keys to tables NG3_BUILDING and NG3_BUILDING_PARTITION, respectively.

Finally, class *Suitability* is mapped to table NG3_SUITABILITY. Column *cityobject_id* contains a foreign key to table NG3_CITYOBJECT. The relation to the child *AbstractSchedule* is provided by the foreign key in column *schedule_id*.

Figure 60 represents graphically the OO-to-ER-model mapping of the Schedule module. Figure 61 depicts the resulting ER-model.

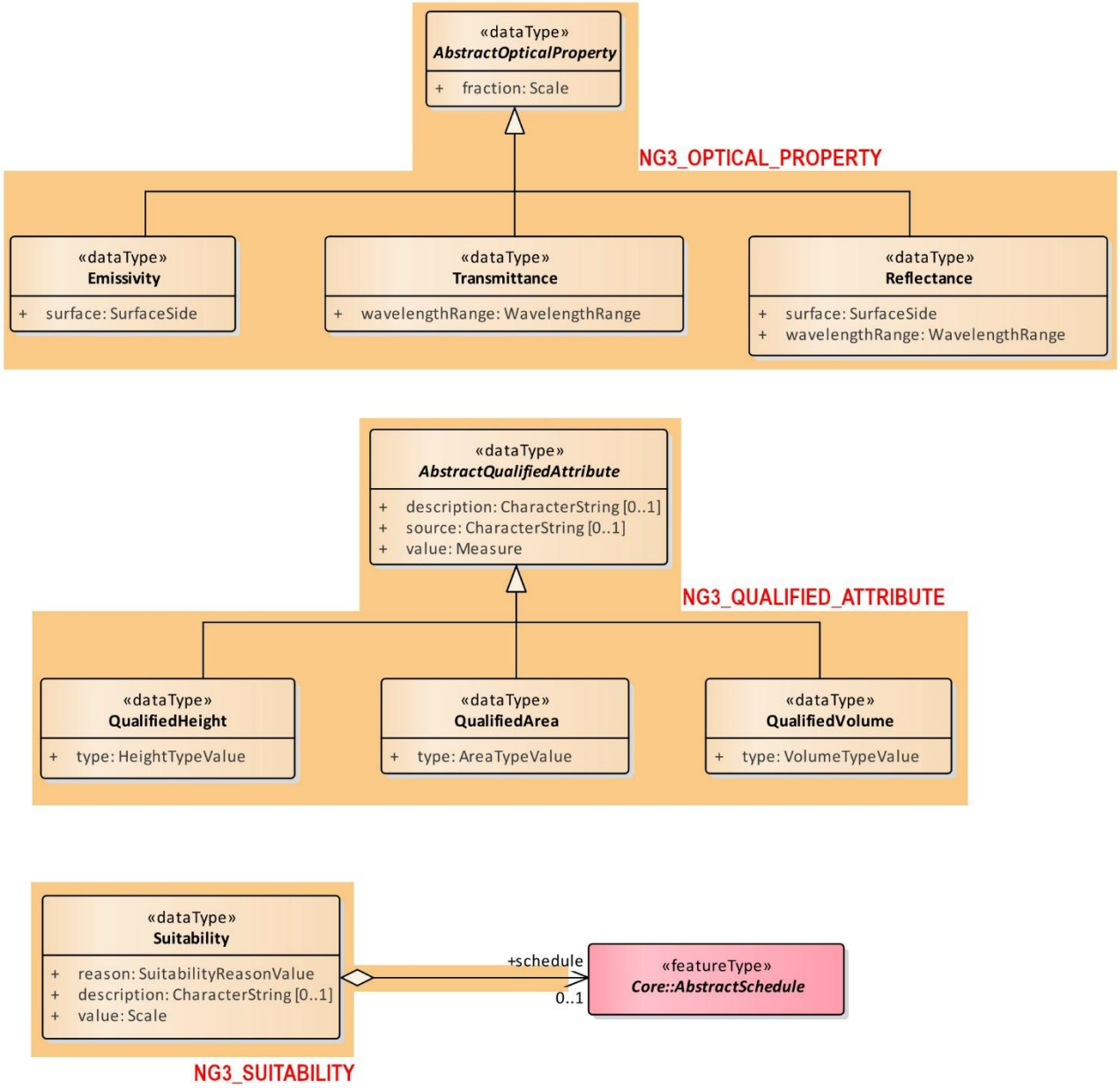


Figure 60. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 DataTypes module.

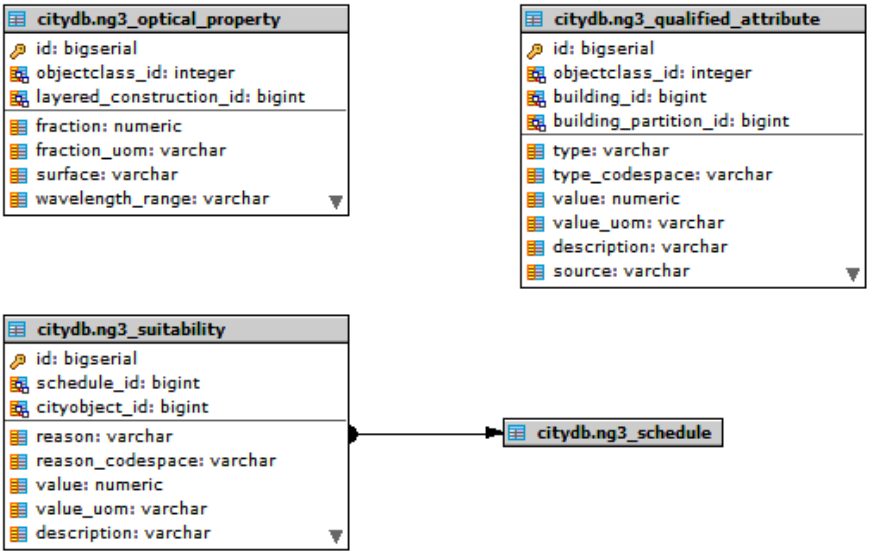


Figure 61. ER-model of the Energy ADE 3.0 DataTypes module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 13, some optical property data are extracted from the 3DCityDB and shown in Figure 62.

```
-- Extract some optical property data
SELECT DISTINCT ON (dt.objectclass_id) dt.id, dt.objectclass_id, o.classname, dt.fraction, dt.fraction_uom, dt.wavelength_range, dt.layered_construction_id
FROM citydb.ng3_optical_property AS dt JOIN citydb.objectclass AS o ON (dt.objectclass_id = o.id)
```

Listing 13. Example of SQL query to extract optical property data from the database.

id bigint	objectclass_id integer	classname character varying (256)	fraction numeric	fraction_uom character varying	wavelength_range character varying	layered_construction_id bigint
20	11002	Emissivity	0.2	unit interval	[null]	476
15	11003	Reflectance	0.2	unit interval	solar	475
32	11004	Transmittance	0.8	unit inteval	solar	[null]

Figure 62. Example of optical property data in the 3DCityDB.

Using the SQL query in Listing 14, some qualified attribute data are extracted from the 3DCityDB and shown in Figure 63. The resulting table is split over multiple lines due to its width.

```
-- Extract some qualified attribute data
SELECT DISTINCT ON (dt.objectclass_id, dt.type) dt.id, dt.objectclass_id, o.classname, dt.type, dt.type_codespace, dt.value, dt.value_uom, dt.source, dt.source, dt.building_id, dt.building_partition_id
FROM citydb.ng3_qualified_attribute AS dt JOIN citydb.objectclass AS o ON (dt.objectclass_id = o.id);
```

Listing 14. Example of SQL query to extract qualified attribute data from the database.

id bigint	objectclass_id integer	classname character varying (256)	type character varying	type_codespace character varying	value numeric	value_uom character va	source character varying
152	11006	QualifiedArea	energyReferenceArea	area_codeSpace	100	m^2	Area value source text
116	11006	QualifiedArea	footprintArea	area_codeSpace	100	m^2	Area value source text
131	11006	QualifiedArea	grossFloorArea	area_codeSpace	250	m^2	Area value source text
16	11007	QualifiedHeight	bottomOfConstruction	height_codeSpace	0	m	Height value source text
33	11007	QualifiedHeight	highestRoofEdge	height_codeSpace	15	m	Height value source text
234	11008	QualifiedVolume	energyReferenceVolume	volume_codeSpace	875	m^3	Volume value source
89	11008	QualifiedVolume	grossVolume	volume_codeSpace	1250	m^3	Volume value source text
30	11008	QualifiedVolume	netVolume	volume_codeSpace	875	m^3	Volume value source text

building_id bigint	building_partition_id bigint
[null]	807
[null]	747
[null]	802
184	[null]
207	[null]
[null]	323
[null]	740
192	[null]

Figure 63. Example of qualified attribute data in the 3DCityDB.

Finally, using the SQL query in Listing 15, some *Suitability* data are extracted from the 3DCityDB and shown in Figure 64.

```
-- Extract some suitability data
SELECT dt.id, dt.reason, dt.reason_codespace, dt.value, dt.value_uom, dt.description, dt.schedule_id, dt.cityobject_id
FROM citydb.ng3_suitability AS dt LIMIT 4;
```

Listing 15. Example of SQL query to extract *Suitability* data from the database.

id [PK] big	reason character varying	reason_codespace character varying	value numeric	value_uom character vary	description character varying	schedule_id bigint	cityobject_id bigint
1	suitability reason 6	suitability_reason_codespace	1	unit interval	suitability description 6	[null]	170
2	suitability reason 14	suitability_reason_codespace	1	unit interval	suitability description 14	[null]	176
3	suitability reason 1	suitability_reason_codespace	0	unit interval	suitability description 1	[null]	184
4	suitability reason 2	suitability_reason_codespace	0.3	unit interval	suitability description 2	[null]	184

Figure 64. Example of *Suitability* data in the 3DCityDB.

7.7.5 Core module

In the Core module, different mapping rules are applied. Following the rules in section 7.6.1, ADE classes *_CityObject* and *_AbstractBuilding* are mapped to the respective target tables NG3_CITYOBJECT and NG3_BUILDING.

Table NG3_CITYOBJECT stores the reference point geometry, while the relation to *AbstractLayeredConstruction* is provided by the foreign key in column *layered_construction_id*. Data of property *suitability* is stored in table NG3_SUITABILITY. The relation *relatedTo* between ADE *_CityObject* and *_CityObject* is stored in table NG3_CTYOBJ_RELATION, which has two columns (*ng3_cityobject_id* and *cityobject_id*) containing foreign keys to table NG3_CITYOBJECT and CITYOBJECT, respectively.

Table NG3_BUILDING stores some simple attributes of ADE *_AbstractBuilding.* Properties such as *height*, *area*, and *volume* are stored in table NG3_QUALIFIED_ATTRIBUTES.

Following the rules in section 7.6.3, class *AbstractLibrary* is mapped to target table NG3_LIBRARY. All attributes up to class *_CityObject* are written in table CITYOBJECT, then all attributes of class *AbstractLibrary* are written in table NG3_LIBRARY. More details are provided in the sections where the specialised classes derived from *AbstractLibrary* are treated (i.e. 7.7.2 and 7.7.3).
Analogously, classes *AbstractBuildingPartition*, *AbstractThermalZone*, *AbstractUsageZone* and *AbstractBuildingUnit* are mapped to one target table NG3_BUILDING_PARTITION. Attributes for *area* and volume are stored in the linked table NG3_QUALIFIED_ATTRIBUTE, while solid geometries are stored in the linked table SURFACE_GEOMETRY, following the default 3DCityDB rules for decomposition of aggregated geometries. More details are provided in the sections covering the modules that make use of class *AbstractBuildingPartition* (i.e. 7.7.8 and 7.7.9).

When it comes to the classes derived directly from class *_Feature* (i.e. *WeatherData*, *DeviceOperation*, *EnergyPerformanceCertificate*, and *RefurbishmentMeasure*), the mapping rules of section 7.6.4 apply. All attributes up to class *_Feature* are written to table CITYOBJECT. Then, all specific attributes are written to linked tables NG3_WEATHER_DATA, NG3_DEVICE_OPERATION, NG3_ENERGY_PERF_CERT, and NG3_REFURBISHMENT_MEASURE, respectively. Table NG3_WEATHER_DATA contains the foreign key to table NG3_CITYOBJECT in column *cityobject_id*, and a foreign key to table NG3_TIME_SERIES in column *time_series_id*. Table NG3_DEVICE_OPERATION contains the foreign key to table NG3_DEVICE in column *device_id* , and a foreign key to table NG3_SCHEDULE in column *schedule_id*. Both tables NG3_ENERGY_PERF_CERT and NG3_REFURBISHMENT_MEASURE contain foreign keys to table NG3_BUILDING and NG3_BUILDING_PARTITION in columns *building_id* and *building_partition_id*, respectively.
The mapping of classes *AbstractTimeSeries* and *AbstractSchedule* is treated in the respective sections 7.7.1 and 7.7.2, respectively.

Figure 65 represents graphically the OO-to-ER-model mapping of the Core module with focus on the class ADE *_CityObject* while Figure 66 and Figure 67 depicts the resulting ER-model. Figure 68 represents graphically the OO-to-ER-model mapping of the Core module with focus on the class ADE *_AbstractBuilding* while Figure 69 depicts the resulting ER-model.

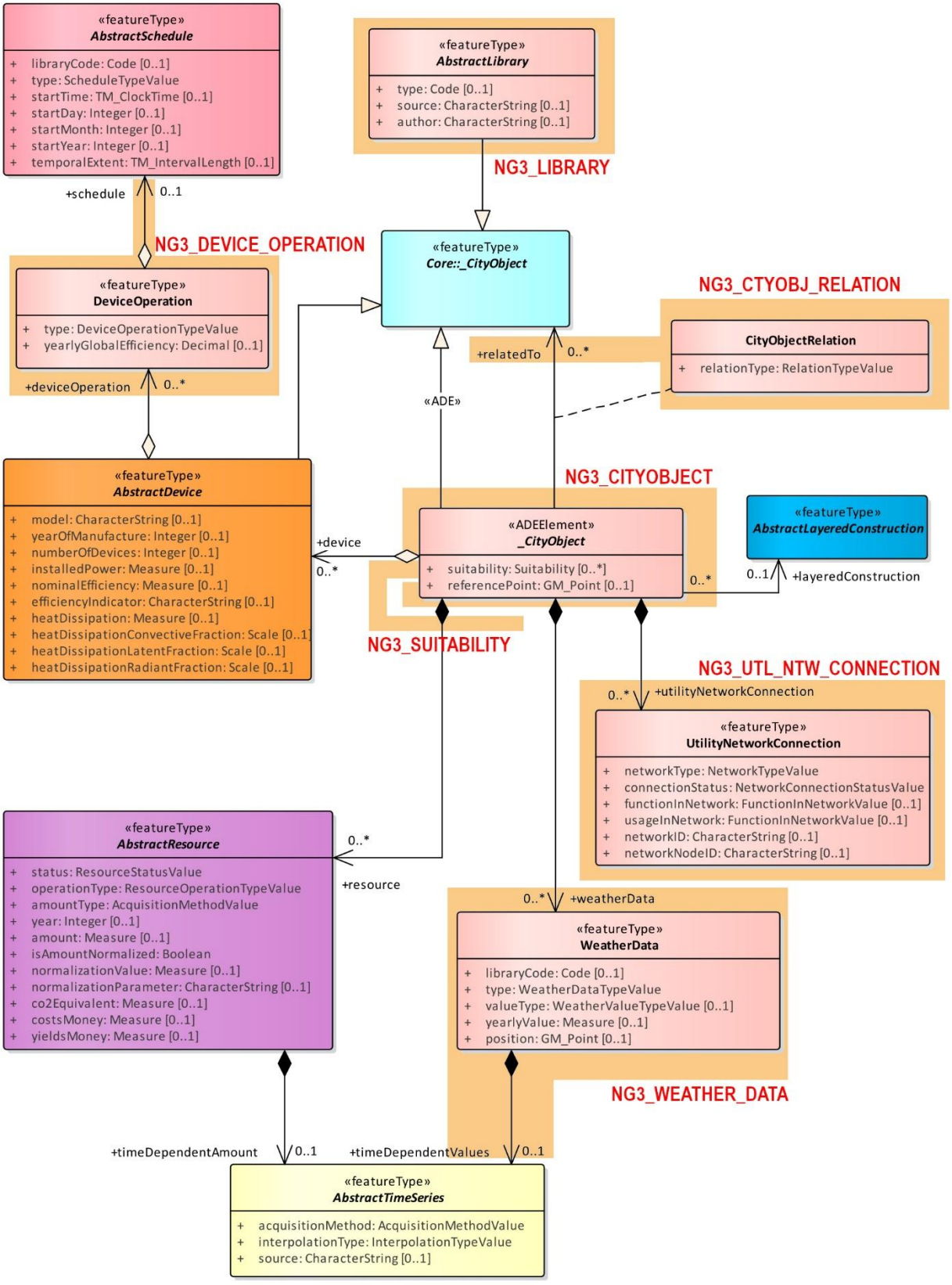


Figure 65. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Core module, focusing on the tables related to class *_CityObject*.

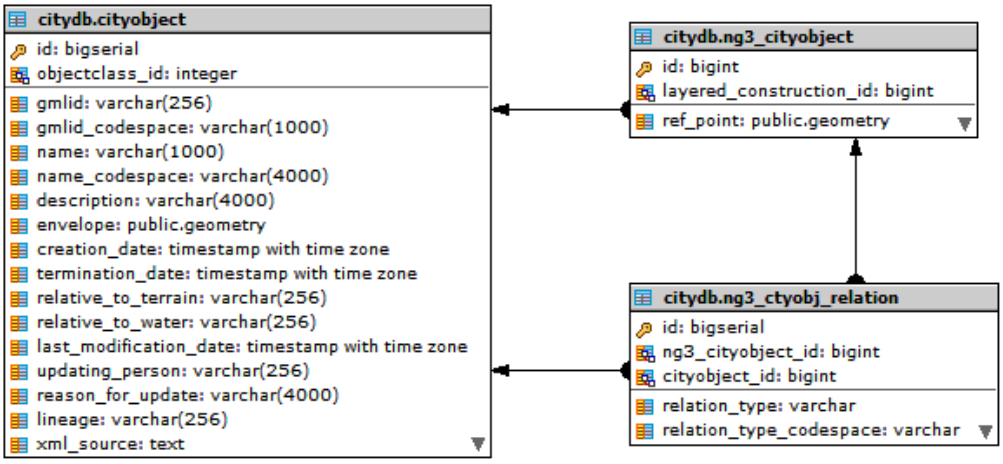


Figure 66. ER-model of the Energy ADE 3.0 Core module, focusing on the tables related to classes ADE _*CityObject* and *CityObjectRelation*.

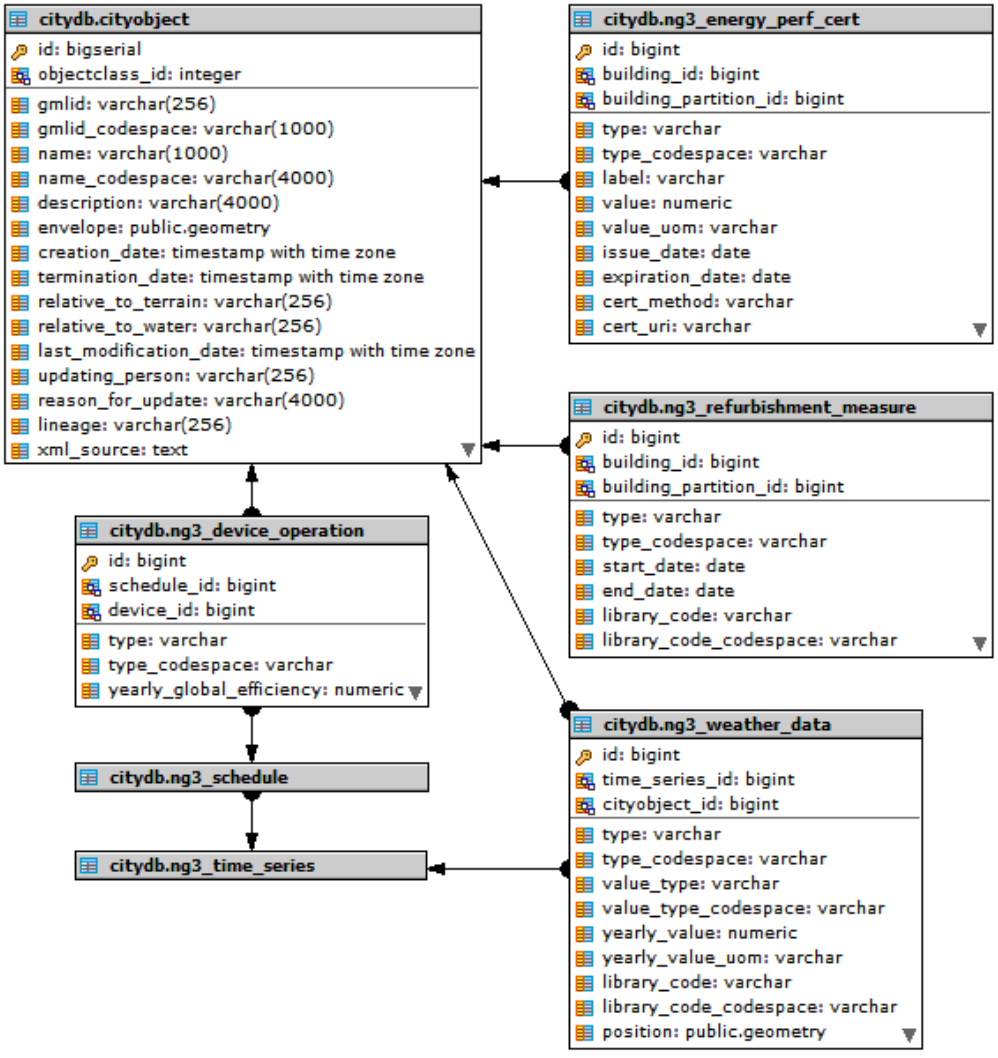


Figure 67. ER-model of the Energy ADE 3.0 Core module, focusing on the tables related to *DeviceOperation*, *EnergyPerformanceCertificate*, *RefubishmentMeasure* and *WeatherData*.

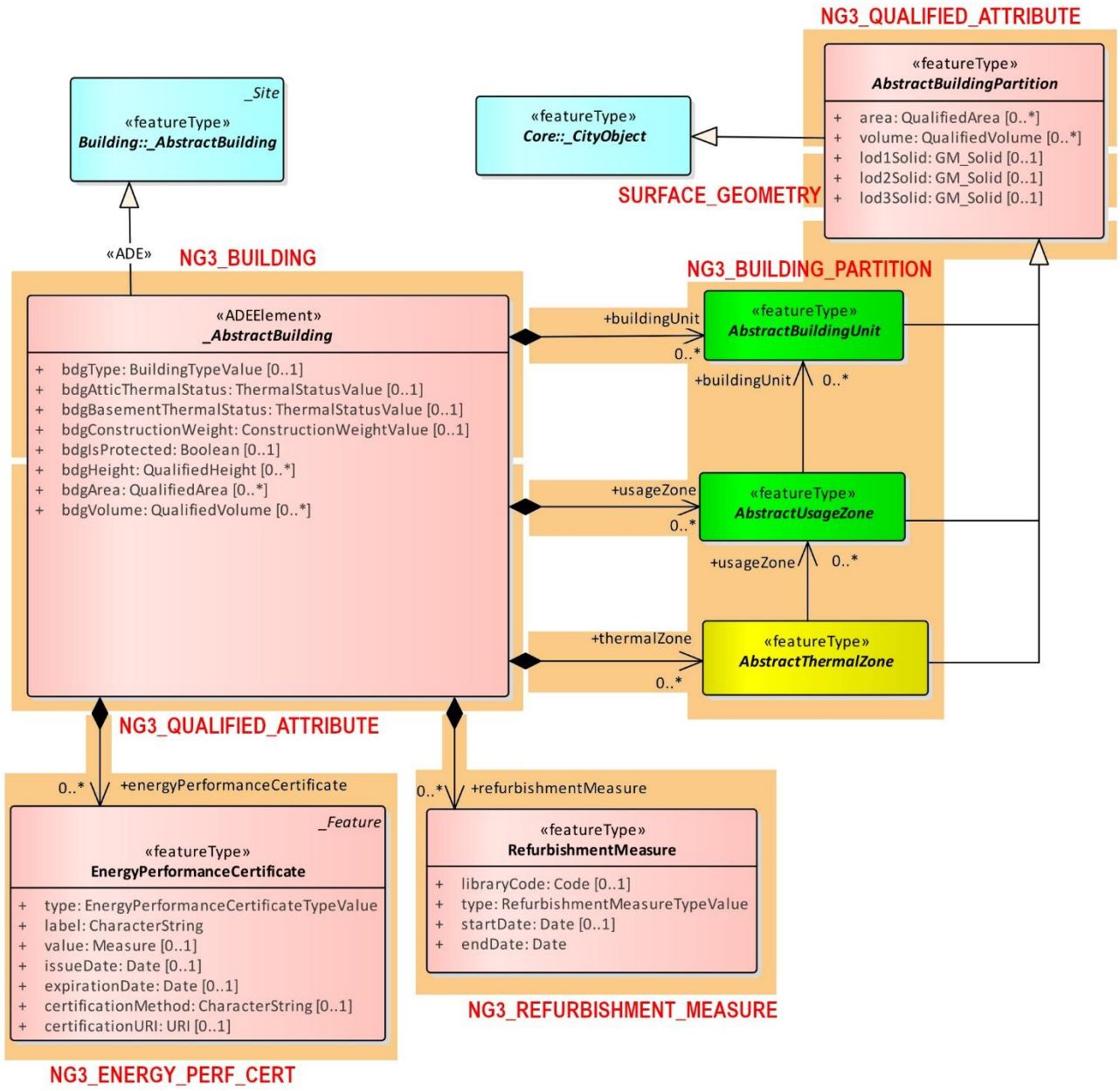


Figure 68. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Core module, focusing on the tables related to class *_AbstractBuilding* and *AbstractBuildingPartition*.

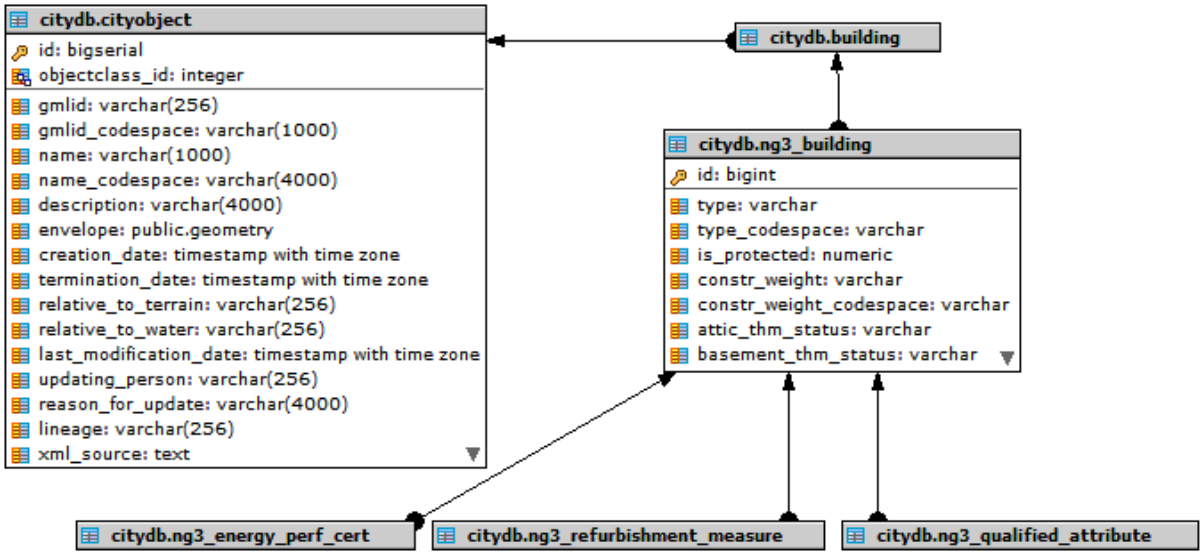


Figure 69. ER-model of the Energy ADE 3.0 Core module, focusing on the tables related to class *_AbstractBuilding*.

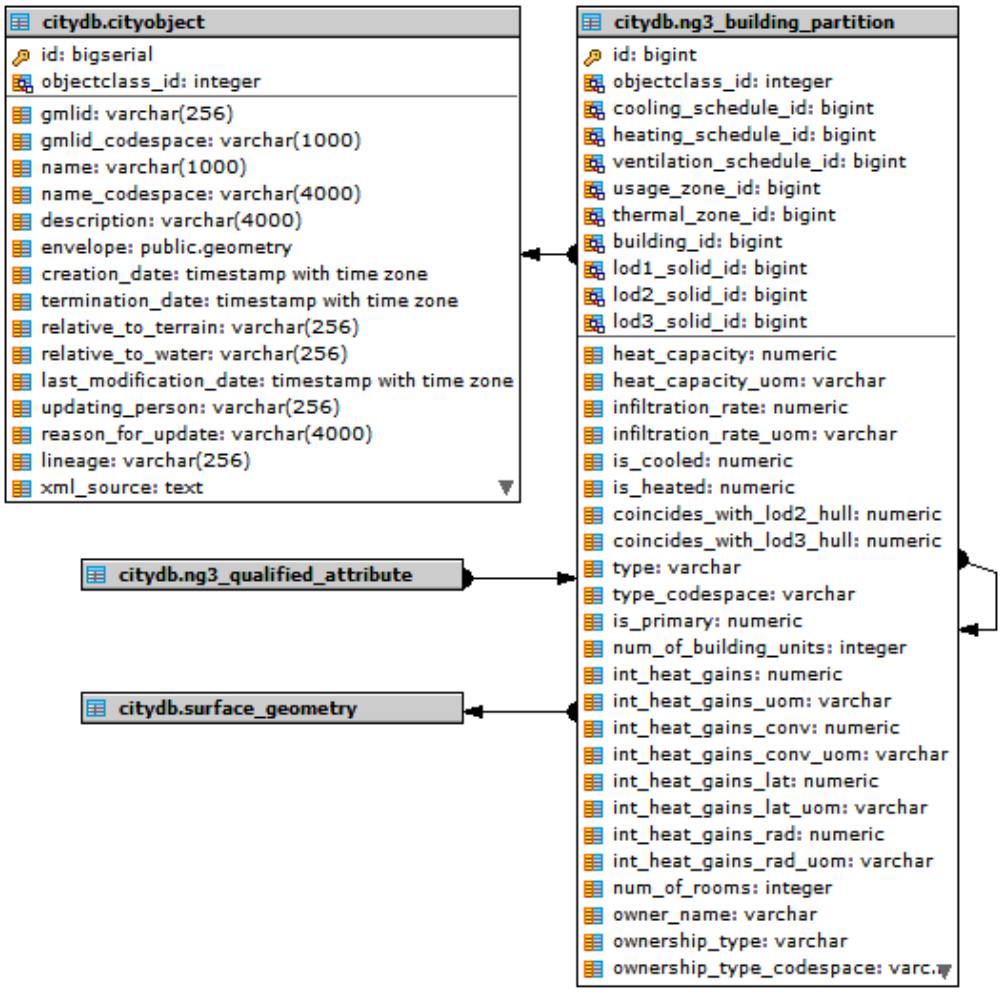


Figure 70. ER-model of the Energy ADE 3.0 Core module, focusing on the tables related to class *AbstractBuildingPartition*.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 16, some data from tables CITYOBJECT and NG3_CITYOBJECT data are extracted from the 3DCityDB and shown in Figure 71. The resulting table is split over multiple lines due to its width.

```
-- Extract some city object data from table CITYOBJECT and NG3_CITYOBJECT
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.* FROM citydb.ng3_cityobject AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname LIMIT 5;
```

Listing 16. Example of SQL query to extract data from tables CITYOBJECT and NG3_CITYOBJECT from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
608	11083	BasementCeilingSurface	id_thermal_zone_11_1_thermal_bo...	[null]	ThermalZone 11_1 ThermalBoundary 4	[null]
480	11102	Boiler	id_boiler_4	[null]	Boiler 4	[null]
176	26	Building	id_building_12	[null]	Death Star II	[null]
526	35	BuildingGroundSurface	id_thermal_zone_1_groundsurface_1	[null]	(ThermalBoundary) GroundSurface 1 (...	[null]
248	25	BuildingPart	id_buildingpart_9	[null]	Jabba's dungeon	[null]

description character varying (4000)	envelope geometry	creation_date timestamp with time	termination_date timestamp with ti	relative_to_terrain character varying (	relative_to_water character varying	id bigint
This is ThermalZone 11_1 ThermalBoundary 4	01030000A0407...	[null]	[null]	[null]	[null]	608
This is Boiler 4	01030000A0407...	[null]	[null]	[null]	[null]	480
This is Building 12	01030000A0407...	2025-05-20 00:10...	[null]	[null]	[null]	176
This is (ThermalBoundary) GroundSurface 1 ...	01030000A0407...	[null]	[null]	[null]	[null]	526
This is BuildingPart 9	01030000A0407...	2025-05-20 00:11...	[null]	[null]	[null]	248

layered_construction_id bigint	ref_point geometry
643	01010000A04...
[null]	01010000A04...
[null]	01010000A04...
474	01010000A04...
[null]	01010000A04...

Figure 71. Example of data from tables CITYOBJECT and NG3_CITYOBJECT in the 3DCityDB.

Using the SQL query in Listing 17, some data from tables CITYOBJECT and NG3_CITYOBJECT data are extracted from the 3DCityDB and shown in Figure 72.

```
-- Extract some CityObjectRelation data
SELECT * FROM citydb.ng3_ctyobj_relation LIMIT 5;
```

Listing 17. Example of SQL query to extract data from tables CITYOBJECT and NG3_CITYOBJECT from the database.

id [PK] big	ng2_cityobject_id bigint	cityobject_id bigint	relation_type character vary	relation_type_codespace character varying
1	170	192	adjacent	adjacent_codeList
2	170	207	adjacent	adjacent_codeList
3	184	207	adjacent	adjacent_codeList
4	191	239	adjacent	adjacent_codeList
5	192	170	adjacent	adjacent_codeList

Figure 72. Example of *CityObjectRelation* data in the 3DCityDB.

Using the SQL query in Listing 18, some WeatherData data are extracted from the 3DCityDB and shown in Figure 73. The resulting table is split over multiple lines due to its width.

```
-- Extract some WeatherData data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'WeatherData' LIMIT 3
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
t.* FROM citydb.ng3_weather_data AS t JOIN co ON (t.id = co.id)
ORDER BY co.id;
```

Listing 18. Example of SQL query to extract *WeatherData* data from the database.

co_id bigint	objectclass_id integer	classname character varyi	gmlid character varying (25	gmlid_codespace character varying (	name character varying	name_codespace character varying	description character varying (4000)	id bigint
878	11043	WeatherData	id_weather_data_1	[null]	WeatherData 1	[null]	This is WeatherData 1	878
879	11043	WeatherData	id_weather_data_2	[null]	WeatherData 2	[null]	This is WeatherData 2	879
880	11043	WeatherData	id_weather_data_3	[null]	WeatherData 3	[null]	This is WeatherData 3	880

type character varying	type_codespace character varying	value_type character va	value_type_codespace character varying	yearly_value numeric	yearly_value_uom character varying	library_code character varyin	library_code_codespace character varying
airTemperature	[null]	minimum	weather_type_codeSpace_1	12	Degrees Celsius	library_code_1	[null]
humidity	[null]	average	weather_type_codeSpace_1	0.6	Scale	library_code_2	[null]
windSpeed	[null]	average	weather_type_codeSpace_1	112	m/s	library_code_3	[null]

time_series_id bigint	cityobject_id bigint	position geometry
762	886	[null]
767	886	[null]
763	887	[null]

Figure 73. Example of *WeatherData* data in the 3DCityDB.

Using the SQL query in Listing 19, some *DeviceOperation* data are extracted from the 3DCityDB and shown in Figure 74. The resulting table is split over multiple lines due to its width.

```
-- Extract some DeviceOperation data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'DeviceOperation'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
t.* FROM citydb.ng3_device_operation AS t JOIN co ON (t.id = co.id)
ORDER BY co.id LIMIT 2;
```

Listing 19. Example of SQL query to extract some *DeviceOperation* data from the database.

co_id bigint	objectclass_id integer	classname character varying (	gmlid character varying (256)	gmlid_codespace character varying	name character varying (100(	name_codespace character varying
331	11100	DeviceOperation	id_operation_schedule_1	[null]	OperationSchedule 1	[null]
332	11100	DeviceOperation	id_operation_schedule_2	[null]	OperationSchedule 2	[null]

name_codespace character varying (	description character varying (4000)	id bigint	type character varyin	type_codespace character varying	yearly_global_efficiency numeric	schedule_id bigint	device_id bigint
[null]	This is OperationSchedule 1	331	spaceHeating	deviceOperation_codeSpace	0.7	427	480
[null]	This is OperationSchedule 2	332	spaceHeating	deviceOperation_codeSpace	0.6	428	481

Figure 74. Example of *DeviceOperation* data in the 3DCityDB.

Using the SQL query in Listing 20, some *EnergyPerformanceCertificate* data are extracted from the 3DCityDB and shown in Figure 75. The resulting table is split over multiple lines due to its width.

```
-- Extract some EnergyPerformanceCertificate data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'EnergyPerformanceCertificate'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
t.* FROM citydb.ng3_energy_perf_cert AS t JOIN co ON (t.id = co.id)
ORDER BY co.id LIMIT 5;
```

Listing 20. Example of SQL query to extract *EnergyPerformanceCertificate* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character var	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
443	11040	EnergyPerformanceCertificate	id_epc_9	[null]	Energy Performance Certificate 9	[null]
444	11040	EnergyPerformanceCertificate	id_epc_10	[null]	Energy Performance Certificate 10	[null]
445	11040	EnergyPerformanceCertificate	id_epc_11	[null]	Energy Performance Certificate 11	[null]
446	11040	EnergyPerformanceCertificate	id_epc_12	[null]	Energy Performance Certificate 12	[null]
447	11040	EnergyPerformanceCertificate	id_epc_13	[null]	Energy Performance Certificate 13	[null]

description character varying (4000)	id bigint	type character v	type_codespace character varying	label charact	value numeric	value_uom character varying	issue_date date	expiration_date date
This is Energy Performance Certificate 9	444	cooling	epc_codeSpace_iop	G	280	kWh/(m^2*a)	2017-01-15	2027-05-23
This is Energy Performance Certificate 10	445	total	epc_codeSpace_iop	B	50	kWh/(m^2*a)	2018-01-15	2027-05-24
This is Energy Performance Certificate 11	446	heating	epc_codeSpace_iop	A	20	kWh/(m^2*a)	2019-01-15	2027-05-25
This is Energy Performance Certificate 12	447	heating	epc_codeSpace_iop	A+++	10	kWh/(m^2*a)	2017-05-15	2027-05-15
This is Energy Performance Certificate 13	448	DHW	epc_codeSpace_iop	B	50	kWh/(m^2*a)	2018-05-15	2027-05-16

cert_method character varying	cert_uri character varying	building_id bigint	building_partition_id bigint
Certification Method 5	http://www.epc.nl	231	[null]
Certification Method 6	http://www.epc.nl	215	[null]
Certification Method 7	http://www.epc.nl	247	[null]
CasaClima	http://www.epc.nl	[null]	314
CasaClima	http://www.epc.nl	[null]	314

Figure 75. Example of *EnergyPerformanceCertificate* data in the 3DCityDB.

Using the SQL query in Listing 21, some *RefurbishmentMeasure* data are extracted from the 3DCityDB and shown in Figure 76. The resulting table is split over multiple lines due to its width.

```
-- Extract some RefurbishmentMeasure data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'RefurbishmentMeasure'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
t.* FROM citydb.ng3_refurbishment_measure AS t JOIN co ON (t.id = co.id)
ORDER BY co.id LIMIT 5;
```

Listing 21. Example of SQL query to extract *RefurbishmentMeasure* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
634	11041	RefurbishmentMeasure	id_refurbishment_10	[null]	Refurbishment Measure 10	[null]
635	11041	RefurbishmentMeasure	id_refurbishment_11	[null]	Refurbishment Measure 11	[null]
636	11041	RefurbishmentMeasure	id_refurbishment_12	[null]	Refurbishment Measure 12	[null]
637	11041	RefurbishmentMeasure	id_refurbishment_13	[null]	Refurbishment Measure 13	[null]
638	11041	RefurbishmentMeasure	id_refurbishment_14	[null]	Refurbishment Measure 14	[null]

description character varying (4000)	id bigint	type character varying	type_codespace character varying	start_date date	end_date date	library_code character varying
This is Refurbishment Measure 10	635	installedSolarPanels	refurb_codeSpace	2020-02-10	2020-08-10	refurb_code_10
This is Refurbishment Measure 11	636	installedPVPanels	refurb_codeSpace	2020-02-11	2020-08-11	refurb_code_11
This is Refurbishment Measure 12	637	other	refurb_codeSpace	2020-02-12	2020-08-12	refurb_code_12
This is Refurbishment Measure 13	638	changedWindows	refurb_codeSpace	2020-02-13	2020-08-13	refurb_code_13
This is Refurbishment Measure 14	639	changedDoors	refurb_codeSpace	2020-02-14	2020-08-14	refurb_code_14

library_code_codespace character varying	building_id bigint	building_partition_id bigint
refurb_lib_codeSpace	215	[null]
refurb_lib_codeSpace	247	[null]
refurb_lib_codeSpace	[null]	314
refurb_lib_codeSpace	[null]	314
refurb_lib_codeSpace	[null]	315

Figure 76. Example of *RefurbishmentMeasure* data in the 3DCityDB.

Using the SQL query in Listing 22, some *Building* and *BuildingPart* data are extracted from the 3DCityDB and shown in Figure 77. The resulting table is split over multiple lines due to its width.

```
-- Extract some Building and BuildingPart data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname IN ('Building', 'BuildingPart')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, b.*, t.*
FROM co JOIN citydb.building AS b ON (co.id = b.id) JOIN citydb.ng3_building AS t ON (b.id = t.id)
ORDER BY co.id LIMIT 4;
```

Listing 22. Example of SQL query to extract some *Building* and *BuildingPart* data from the database.

co_id bigint	objectclass_id integer	classname character varyi	gmlid character varying (25	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying (	description character varying (4000)	envelope geometry	creation_date timestamp with
184	26	Building	id_building_1	[null]	Snoke's Palace	[null]	This is Building 1	01030000A0407...	2025-05-20 ...
247	26	Building	id_building_9-10	[null]	Jabba's multi-part Palace	[null]	This is multi-part Building 9-10	01030000A0407...	2025-05-20 ...
248	25	BuildingPart	id_buildingpart_9	[null]	Jabba's dungeon	[null]	This is BuildingPart 9	01030000A0407...	2025-05-20 ...
256	25	BuildingPart	id_buildingpart_10	[null]	The rancor's prison	[null]	This is BuildingPart 10	01030000A0407...	2025-05-20 ...

termination_date timestamp with ti	relative_to_terrain character varying (	relative_to_water character varying (	id bigint	objectclass_id integer	building_parent_id bigint	building_root_id bigint	class character var	class_codespace character varying (4000)	function character varying (10
[null]	[null]	[null]	184	26	[null]	184	habitation	http://www.sig3d.org/codelists/standar...	residential building
[null]	[null]	[null]	247	26	[null]	247	habitation	http://www.sig3d.org/codelists/standar...	residential building
[null]	[null]	[null]	248	25	247	247	habitation	http://www.sig3d.org/codelists/standar...	residential building
[null]	[null]	[null]	256	25	247	247	habitation	http://www.sig3d.org/codelists/standar...	residential building

function_codespace character varying (4000)	usage character varying (1000)	usage_codespace character varying (4000)	year_of_construction date	year_of_demolition date	roof_type character varying (256)	roof_type_codespace character varying (4000)
http://www.sig3d.org/codelist...	[null]	[null]	1955-01-01	[null]	gabled roof	http://www.sig3d.org/codelists/...
http://www.sig3d.org/codelist...	[null]	[null]	1965-01-01	[null]	gabled roof	http://www.sig3d.org/codelists/...
http://www.sig3d.org/codelist...	[null]	[null]	1965-01-01	[null]	gabled roof	http://www.sig3d.org/codelists/...
http://www.sig3d.org/codelist...	[null]	[null]	1940-01-01	[null]	gabled roof	http://www.sig3d.org/codelists/...

measured_height double precision	measured_height_unit character varying (4000)	storeys_above_ground numeric (8)	storeys_below_ground numeric (8)	storey_heights_above_ground character varying (4000)	storey_heights_ag_unit character varying (4000)	storey_heights_below_ground character varying (4000)	storey_heights_bg_unit character varying (4000)
15	m	3	0	3.0	m	[null]	[null]
15	m	3	5	3.0	m	[null]	[null]
15	m	3	5	3.0	m	[null]	[null]
15	m	3	0	3.0	m	[null]	[null]

lod1_terrain_int geometry	lod2_terrain_int geometry	lod3_terrain_int geometry	lod4_terrain_int geometry	lod2_multi_c geometry	lod3_multi_c geometry	lod4_multi_c geometry	lod0_footprint_id bigint	lod0_roofprint_id bigint	lod1_multi_surface_id bigint
[null]	[null]	[null]	[null]	[null]	[null]	[null]	1228	[null]	[null]
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
[null]	[null]	[null]	[null]	[null]	[null]	[null]	1657	[null]	[null]
[null]	[null]	[null]	[null]	[null]	[null]	[null]	1711	[null]	[null]

lod2_multi_surface_id bigint	lod3_multi_surface_id bigint	lod4_multi_surface_id bigint	lod1_solid_id bigint	lod2_solid_id bigint	lod3_solid_id bigint	lod4_solid_id bigint	id bigint	type character varying	type_codespace character varying	is_protected numeric	constr_weight character varyi
[null]	[null]	[null]	1231	1245	[null]	[null]	184	terracedHouse	bdgType_codeSpace	0	heavy
[null]	[null]	[null]	[null]	[null]	[null]	[null]	247	[null]	[null]	0	heavy
[null]	[null]	[null]	1660	1674	[null]	[null]	248	terracedHouse	bdgType_codeSpace	0	light
[null]	[null]	[null]	1714	1728	[null]	[null]	256	terracedHouse	bdgType_codeSpace	0	light

constr_weight_codespace character varying	attic_thm_status character varying	basement_thm_status character varying
constrWeight_codeSpace	isOnlyCooled	isOnlyHeated
constrWeight_codeSpace	isHeatedAnd...	isNeitherHeatedNo...
constrWeight_codeSpace	isOnlyHeated	isOnlyHeated
constrWeight_codeSpace	isOnlyHeated	isOnlyHeated

Figure 77. Example of *Building* and *BuildingPart* data in the 3DCityDB.

7.7.6 Weather station module

Following the rules in section 7.6.3, class *WeatherStation* is simply mapped to table CITYOBJECT. Figure 78 represents graphically the OO-to-ER-model mapping of the Weather station module. Figure 79 depicts the resulting ER-model.

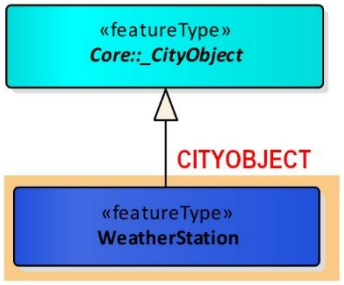


Figure 78. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Weather station module.

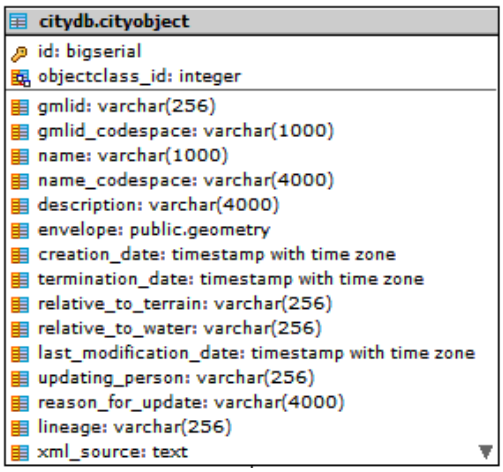


Figure 79. ER-model of the Energy ADE 3.0 Weather station module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 23, some *WeatherStation* data are extracted from the 3DCityDB and shown in Figure 80. The resulting table is split over multiple lines due to its width.

```
-- Extract WeatherStation data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM citydb.ng3_cityobject AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname LIMIT 10;
```

Listing 23. Example of SQL query to extract *WeatherStation* data from the database.

co_id bigint	objectclass_id integer	classname character varying	gmlid character varying (256)	gmlid_codespace character varying (1	name character varying (1	name_codespace character varying (
886	11130	WeatherStation	id_weather_station_1	[null]	Weather Station 1	[null]
887	11130	WeatherStation	id_weather_station_2	[null]	Weather Station 2	[null]

description character varying (4000)	envelope geometry	creation_date timestamp with	termination_date timestamp with tin	relative_to_terrain character varying (256)	relative_to_water character varying (
This is Weather Station 1	01030000A040710…	[null]	[null]	entirelyAboveTerrain	[null]
This is Weather Station 2	01030000A040710…	[null]	[null]	entirelyAboveTerrain	[null]

Figure 80. Example of *WeatherStation* data in the 3DCityDB.

7.7.7 Urban function area module

Following the rules in section 7.6.2, class *UrbanFunctionArea* is mapped to linked table NG3_URBAN_FUNCTION_AREA. All attributes up to class *_CityObject* are written to table CITYOBJECT, then all properties inherited from class *CityObjectGroup* are written to table CITYOBJECTGROUP and, if needed, ancillary tables GROUP_TO_CITYOBJECT and SURFACE_GEOMETRY. Finally, table NG3_URBAN_FUNCTION_AREA contains the attributes of class *UrbanFunctionArea*.

Figure 81 represents graphically the OO-to-ER-model mapping of the Weather station module. Figure 82 depicts the resulting ER-model.

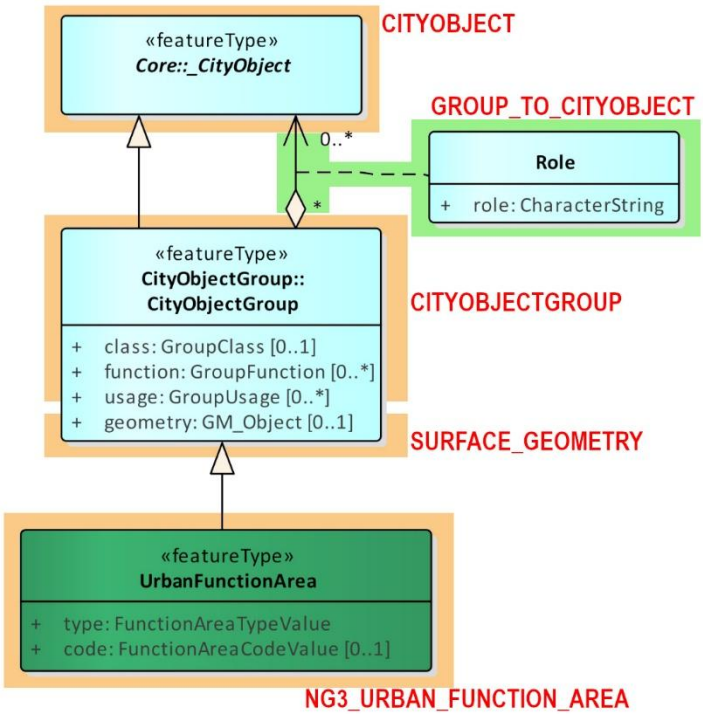


Figure 81. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Urban function area module

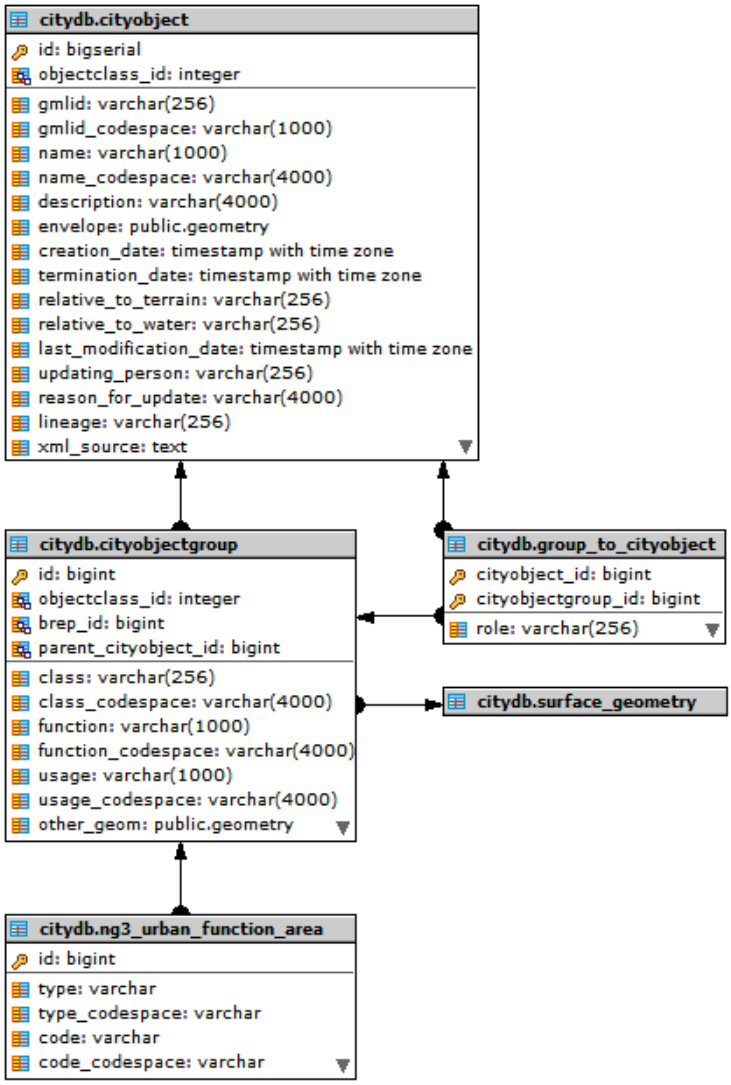


Figure 82. ER-model of the Energy ADE 3.0 Urban function area module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 24, some *UrbanFunctionArea* data are extracted from the 3DCityDB and shown in Figure 83. The resulting table is split over multiple lines due to its width.

```
-- Extract UrbanFunctionArea data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM citydb.ng3_cityobject AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname LIMIT 10;
```

Listing 24. Example of SQL query to extract *UrbanFunctionArea* data from the database.

co_id bigint	objectclass_id integer	classname character varying (25	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying (
771	11120	UrbanFunctionArea	id_urban_function_area_1	[null]	UrbanFunctionArea of first level	[null]
772	11120	UrbanFunctionArea	id_urban_function_area_1_1	[null]	UrbanFunctionArea of second level	[null]
773	11120	UrbanFunctionArea	id_urban_function_area_1_1_1	[null]	UrbanFunctionArea of third level	[null]
774	11120	UrbanFunctionArea	id_urban_function_area_1_1_2	[null]	UrbanFunctionArea of third level	[null]

description character varying (4000)	envelope geometry	creation_date timestamp with	termination_date timestamp with ti	relative_to_terrain character varying (	relative_to_water character varying	id bigint
This is an UrbanFunctionArea of first level	01030000A0407...	[null]	[null]	[null]	[null]	771
This is an UrbanFunctionArea of second level	01030000A0407...	[null]	[null]	[null]	[null]	772
This is an UrbanFunctionArea of third level	01030000A0407...	[null]	[null]	[null]	[null]	773
This is an UrbanFunctionArea of third level	01030000A0407...	[null]	[null]	[null]	[null]	774

objectclass_id integer	class character varying (256)	class_codespace character varying (	function character varying (1000)	function_codespace character varying (40	usage character varying (1000)
11120	UrbanFuncionArea_class	[null]	UrbanFuncionArea_function	[null]	UrbanFuncionArea_usage
11120	UrbanFuncionArea_class	[null]	UrbanFuncionArea_function	[null]	UrbanFuncionArea_usage
11120	UrbanFuncionArea_class	[null]	UrbanFuncionArea_function	[null]	UrbanFuncionArea_usage
11120	UrbanFuncionArea_class	[null]	UrbanFuncionArea_function	[null]	UrbanFuncionArea_usage

usage_codespace character varying (	brep_id bigint	other_geom geometry	parent_cityobject_id bigint	id bigint	type character varying	type_codespace character varying
[null]	2627	[null]	[null]	771	urbfuncarea_type_1	urbfuncarea_type_codeSpace
[null]	2629	[null]	[null]	772	urbfuncarea_type_12	urbfuncarea_type_codeSpace
[null]	2631	[null]	[null]	773	urbfuncarea_type_13	urbfuncarea_type_codeSpace
[null]	2633	[null]	[null]	774	urbfuncarea_type_13	urbfuncarea_type_codeSpace

code character varying	code_codespace character varying
urbfuncarea_code_1	urbfuncarea_code_codeSpace
urbfuncarea_code_12	urbfuncarea_code_codeSpace
urbfuncarea_code_13	urbfuncarea_code_codeSpace
urbfuncarea_code_13	urbfuncarea_code_codeSpace

Figure 83. Example of *UrbanFunctionArea* data in the 3DCityDB.

7.7.8 Occupancy module

In the Occupancy module, different mapping rules are applied. Following the rules in section 7.6.3 and as already described in section 7.7.5, classes *UsageZone* and *BuildingPartition* (and their parent class *AbstractBuildingPartition* from Core module) are mapped to just one target table NG3_BUILDING_PARTITION. Please note that also class *ThermalZone* (from Building physics module) is mapped to the same table, as described in section 7.7.9. This means that, depending on the specific class, some columns will be written, while others may remain empty.

For class *UsageZone*, the relations to *AbstractSchedule* for heating, cooling and ventilation are realised by means of foreign keys in columns *heating_schedule_id*, *cooling_schedule_id*, and *ventilation_schedule_id*, respectively. For class *BuildingUnit*, the relation to *Address* is realised by means of the association table NG3_ADDRESS_TO_BUILDING_UNIT.

EnergyPerformanceCertificate and *RefurbishmentMeasure* data are stored in the linked tables NG3_ENERGY_PERF_CERT and NG3_REFURBISHMENT_MEASURE, respectively.

Class *Occupants* is mapped following the rules of section 7.6.4. All attributes up to class *_Feature* are written to table CITYOBJECT. Then, all specific attributes are written to table NG3_OCCUPANTS. Both relations *occupiedBy* are realised by means of the foreign key in column *building_partition_id*.

Figure 84 represents graphically the OO-to-ER-model mapping of the Occupancy module while Figure 85 depicts the resulting ER-model.

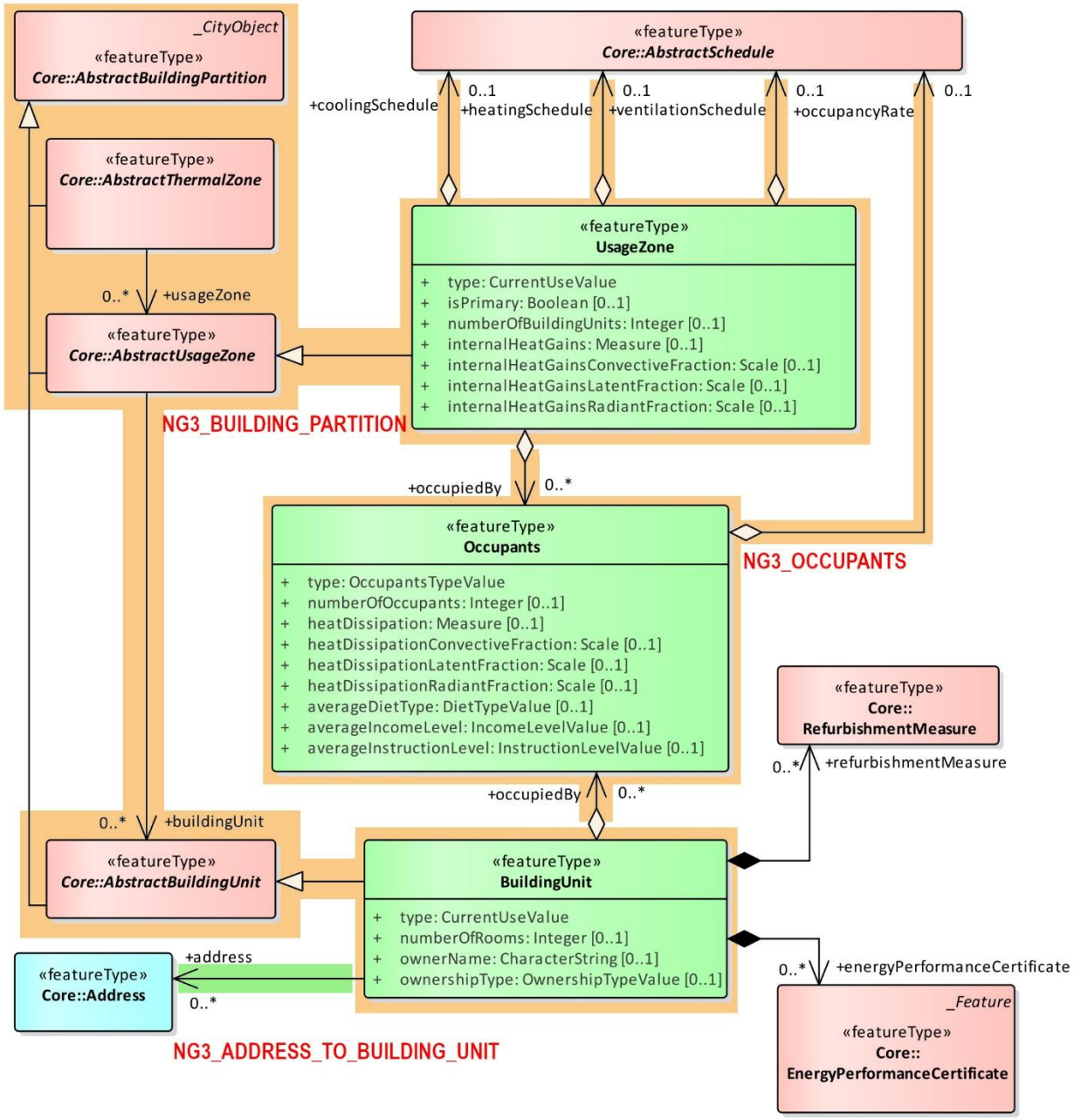
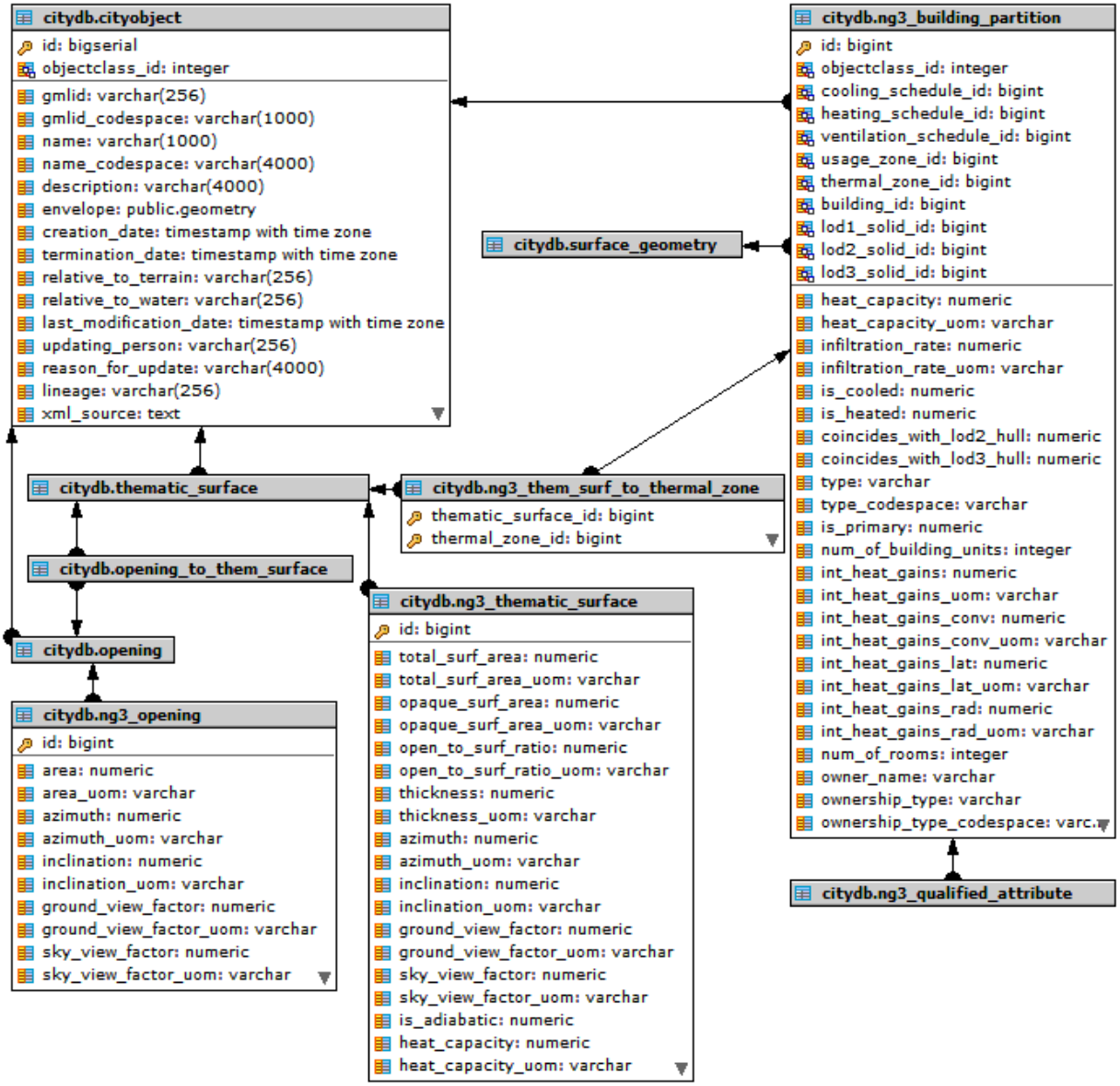


Figure 84. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 2.Occupancy module.


citydb.cityobject
id: bigserial
objectclass_id: integer
gmlid: varchar(256)
gmlid_codespace: varchar(1000)
name: varchar(1000)
name_codespace: varchar(4000)
description: varchar(4000)
envelope: public.geometry
creation_date: timestamp with time zone
termination_date: timestamp with time zone
relative_to_terrain: varchar(256)
relative_to_water: varchar(256)
last_modification_date: timestamp with time zone
updating_person: varchar(256)
reason_for_update: varchar(4000)
lineage: varchar(256)
xml_source: text
citydb.ng3_building_partition
id: bigint
objectclass_id: integer
cooling_schedule_id: bigint
heating_schedule_id: bigint
ventilation_schedule_id: bigint
usage_zone_id: bigint
thermal_zone_id: bigint
building_id: bigint
lod1_solid_id: bigint
lod2_solid_id: bigint
lod3_solid_id: bigint
heat_capacity: numeric
heat_capacity_uom: varchar
infiltration_rate: numeric
infiltration_rate_uom: varchar
is_cooled: numeric
is_heated: numeric
coincides_with_lod2_hull: numeric
coincides_with_lod3_hull: numeric
type: varchar
type_codespace: varchar
is_primary: numeric
num_of_building_units: integer
int_heat_gains: numeric
int_heat_gains_uom: varchar
int_heat_gains_conv: numeric
int_heat_gains_conv_uom: varchar
int_heat_gains_lat: numeric
int_heat_gains_lat_uom: varchar
int_heat_gains_rad: numeric
int_heat_gains_rad_uom: varchar
num_of_rooms: integer
owner_name: varchar
ownership_type: varchar
ownership_type_codespace: varc...
citydb.surface_geometry
citydb.thematic_surface
citydb.ng3_them_surf_to_thermal_zone
thematic_surface_id: bigint
thermal_zone_id: bigint
citydb.opening_to_them_surface
citydb.ng3_thematic_surface
id: bigint
total_surf_area: numeric
total_surf_area_uom: varchar
opaque_surf_area: numeric
opaque_surf_area_uom: varchar
open_to_surf_ratio: numeric
open_to_surf_ratio_uom: varchar
thickness: numeric
thickness_uom: varchar
azimuth: numeric
azimuth_uom: varchar
inclination: numeric
inclination_uom: varchar
ground_view_factor: numeric
ground_view_factor_uom: varchar
sky_view_factor: numeric
sky_view_factor_uom: varchar
is_adiabatic: numeric
heat_capacity: numeric
heat_capacity_uom: varchar
citydb.opening
citydb.ng3_opening
id: bigint
area: numeric
area_uom: varchar
azimuth: numeric
azimuth_uom: varchar
inclination: numeric
inclination_uom: varchar
ground_view_factor: numeric
ground_view_factor_uom: varchar
sky_view_factor: numeric
sky_view_factor_uom: varchar
citydb.ng3_qualified_attribute

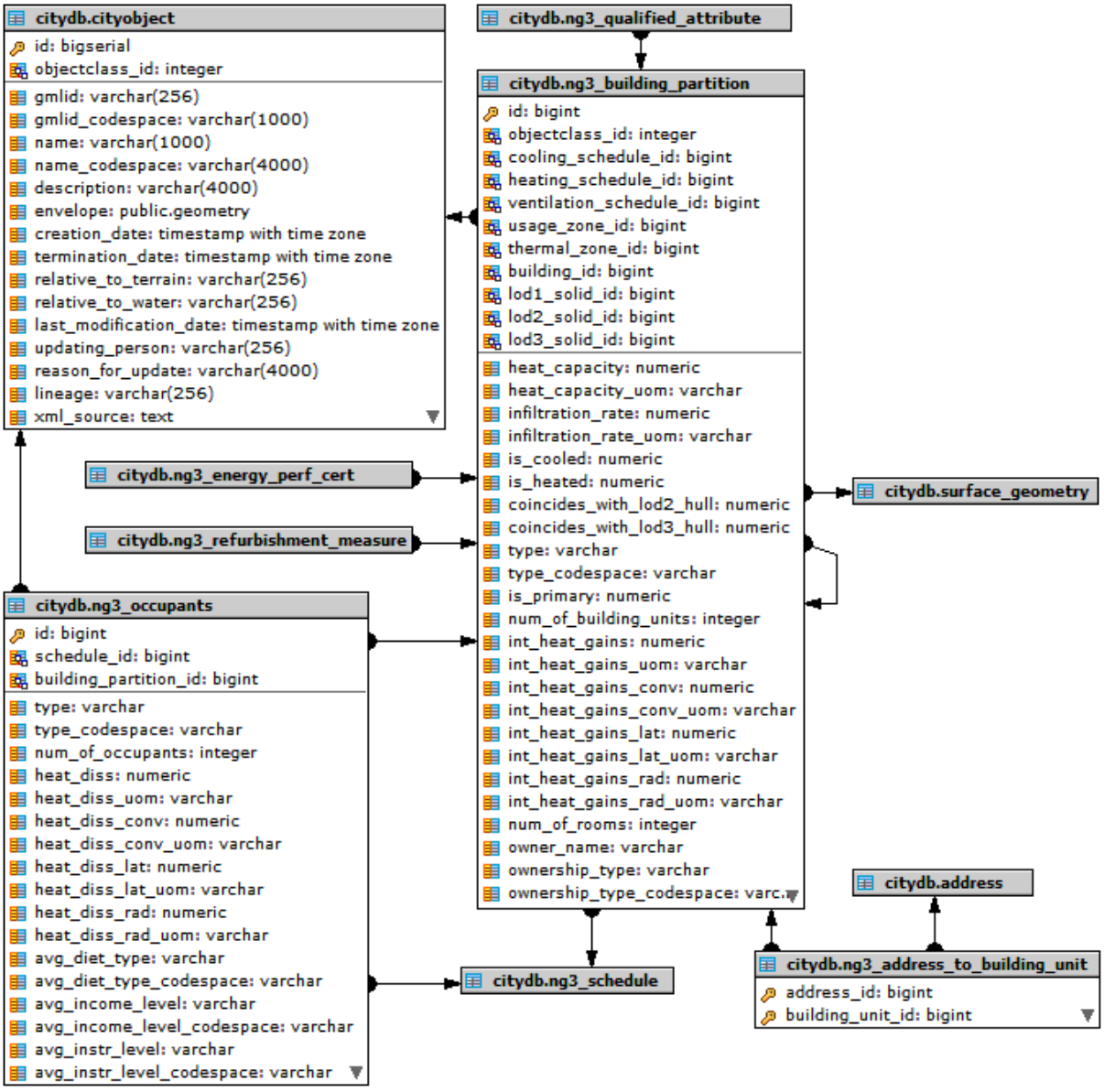


Figure 85. ER-model of the Energy ADE 3.0 Occupancy module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 25, some *UsageZone* and some *BuildingUnit* data are extracted from the 3DCityDB and shown in Figure 86. In this example, the query extracts data of both classes, as they are stored in the same tables. The resulting table is split over multiple lines due to its width.

```
-- Extract UsageZone and BuildingUnit data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname IN ('UsageZone', 'BuildingUnit')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM co JOIN citydb.ng3_building_partition AS t ON (co.id = t.id)
ORDER BY co.id LIMIT 4;
```

Listing 25. Example of SQL query to extract *UsageZone* and *BuildingUnit* data from the database.

co_id bigint	objectclass_id integer	classname character varyi	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1	name_codespace character varying (	description character varying (4000)	envelope geometry	creation_date timestamp wit	termination_date timestamp with ti
329	11093	BuildingUnit	id_building_unit_11_2	[null]	BuildingUnit 11_2	[null]	This is BuildingUnit 11_2	01030000A04...	[null]	[null]
330	11093	BuildingUnit	id_building_unit_12	[null]	BuildingUnit 12	[null]	This is BuildingUnit 12	01030000A04...	[null]	[null]
802	11091	UsageZone	id_usage_zone_1	[null]	UsageZone 1	[null]	This is UsageZone 1	01030000A04...	[null]	[null]
803	11091	UsageZone	id_usage_zone_2_1	[null]	UsageZone 2_1	[null]	This is UsageZone 2_1	01030000A04...	[null]	[null]

termination_date timestamp with ti	relative_to_terrain character varying (2	relative_to_water character varying (	id bigint	objectclass_id integer	heat_ca numeric	heat_ca charact	infiltrati numeric	infiltrati charact	is_coole numeric	is_heate numeric	coincide numeric	coincide numeric	type character va	type_codespace character varying	is_primary numeric
[null]	[null]	[null]	329	11093	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	residential	[null]	[null]
[null]	[null]	[null]	330	11093	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	residential	[null]	[null]
[null]	[null]	[null]	802	11091	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	residential	[null]	1
[null]	[null]	[null]	803	11091	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	residential	[null]	1

Unused columns.
Required only for *ThermalZone* data

num_of_building_units integer	int_heat_gains numeric	int_heat_gains_uom character varying	int_heat_gains_conv numeric	int_heat_gains_conv_uom character varying	int_heat_gains_lat numeric	int_heat_gains_lat_uom character varying	int_heat_gains_rad numeric
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
1	100	W/m^2	0.3	unit interval	0.2	unit interval	0.5
1	100	W/m^2	0.3	unit interval	0.2	unit interval	0.5

int_heat_gains_rad_uom character varying	num_of_rooms integer	owner_name character varying	ownership_type character varying	ownership_type_codespace character varying	cooling_schedule_id bigint	heating_schedule_id bigint	ventilation_schedule_id bigint
[null]	5	[null]	propertyManagementCompany	ownership_type_codeSpace_abc	[null]	[null]	[null]
[null]	5	[null]	propertyManagementCompany	ownership_type_codeSpace_abc	[null]	[null]	[null]
unit interval	[null]	[null]	[null]	[null]	413	427	424
unit interval	[null]	[null]	[null]	[null]	413	427	424

usage_zone_id bigint	thermal_zone_id bigint	building_id bigint	lod1_solid_id bigint	lod2_solid_id bigint	lod3_solid_id bigint
[null]	[null]	223	3477	3485	3494
[null]	[null]	176	3503	3511	3520
[null]	738	184	2689	2697	2706
[null]	739	207	2715	2723	2731

Figure 86. Example of *UsageZone* and *BuildingUnit* data in the 3DCityDB.

Finally, using the SQL query in Listing 26, some *Occupancy* data are extracted from the 3DCityDB and shown in Figure 87. The resulting table is split over multiple lines due to its width.

```
-- Extract Occupancy data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON
(co.objectclass_id = o.id)
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM citydb.ng3_cityobject AS t JOIN co ON (t.id = co.id)
ORDER BY co.classname LIMIT 10;
```

Listing 26. Example of SQL query to extract *Occupancy* data from the database

co_id bigint	objectclass_id integer	classname character vary	gmlid character varying	gmlid_codespace character varying	name character varyi	name_codespace character varying (·	description character varying (400
499	11094	Occupants	id_occupants_1	[null]	Occupants 1	[null]	This is Occupants 1
500	11094	Occupants	id_occupants_2	[null]	Occupants 2	[null]	This is Occupants 2

id bigint	type character v	type_codespace character varying	num_of_occupants integer	heat_diss numeric	heat_diss_uom character varyin	heat_diss_conv numeric	heat_diss_conv_uom character varying	heat_diss_lat numeric
499	residents	occ_codeSpace_xyz	12	100	W/m^2	0.3	unit interval	0.2
500	patients	occ_codeSpace_xyz	11	100	W/m^2	0.3	unit interval	0.2

heat_diss_lat_uom character varying	heat_diss_rad numeric	heat_diss_rad_uom character varying	avg_diet_type character varyi	avg_diet_type_codespace character varying	avg_income_level character varying	avg_income_level_codespace character varying
unit interval	0.5	unit interval	omnivorous	diet_codeSpace	middle	income_level codeSpace
unit interval	0.5	unit interval	omnivorous	diet_codeSpace	middle	income_level codeSpace

avg_instr_level character varyi	avg_instr_level_codespace character varying	schedule_id bigint	building_partition_id bigint
university	instruction_level_codeS...	414	802
university	instruction_level_codeS...	414	803

Figure 87. Example of *Occupancy* data in the 3DCityDB.

7.7.9 Building physics module

In the Building physics module, different mapping rules are applied. Following the rules in section 7.6.3 and as already described in section 7.7.5, class *ThermalZone* (and its parent class *AbstractBuildingPartition* from Core module) are mapped to just one target table NG3_BUILDING_PARTITION. Please note that also classes *UsageZone* and *BuildingPartition* (from Occupancy module) are mapped to the same table, as described in section 7.7.8. This means that, depending on the specific class, some columns will be written, while others may remain empty

Following the rules in section 7.6.1, classes ADE *_BoundarySurface* and ADE *_Opening* are mapped to the respective linked tables NG3_THEMATIC_SURFACE and NG3_OPENING.
Finally, following the rules in section 7.6.2, all new thematic surface classes (*AtticFloorSurface*, *PartyWallSurface*, etc.) are mapped to tables CITYOBJECT and THEMATIC_SURFACE.
The relation *thermalBoundary* is realised by means of the association table NG3_THEM_SURF_TO_THERMAL_ZONE.
Figure 88 represents graphically the OO-to-ER-model mapping of the Building physics module while Figure 89 depicts the resulting ER-model.

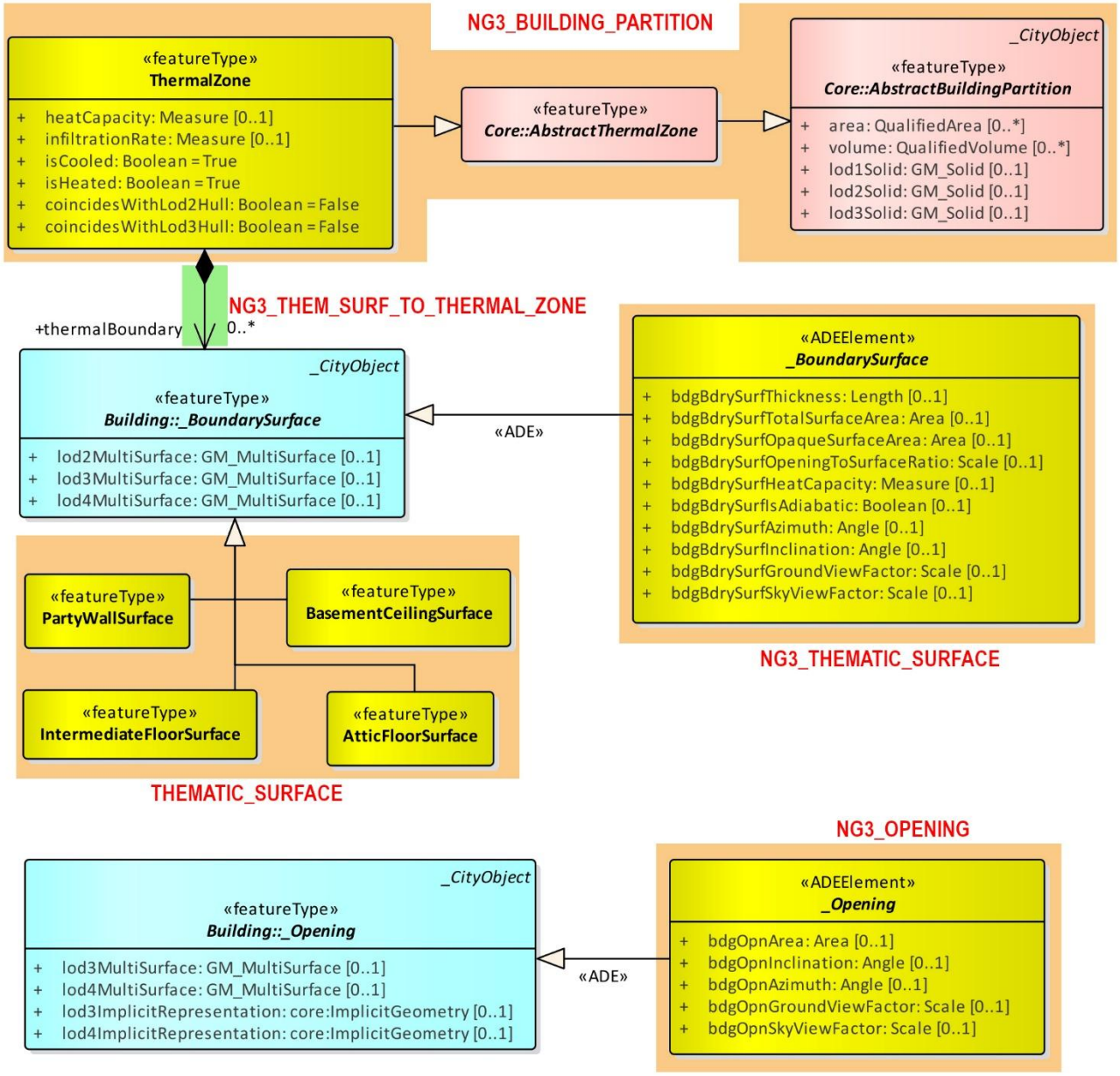
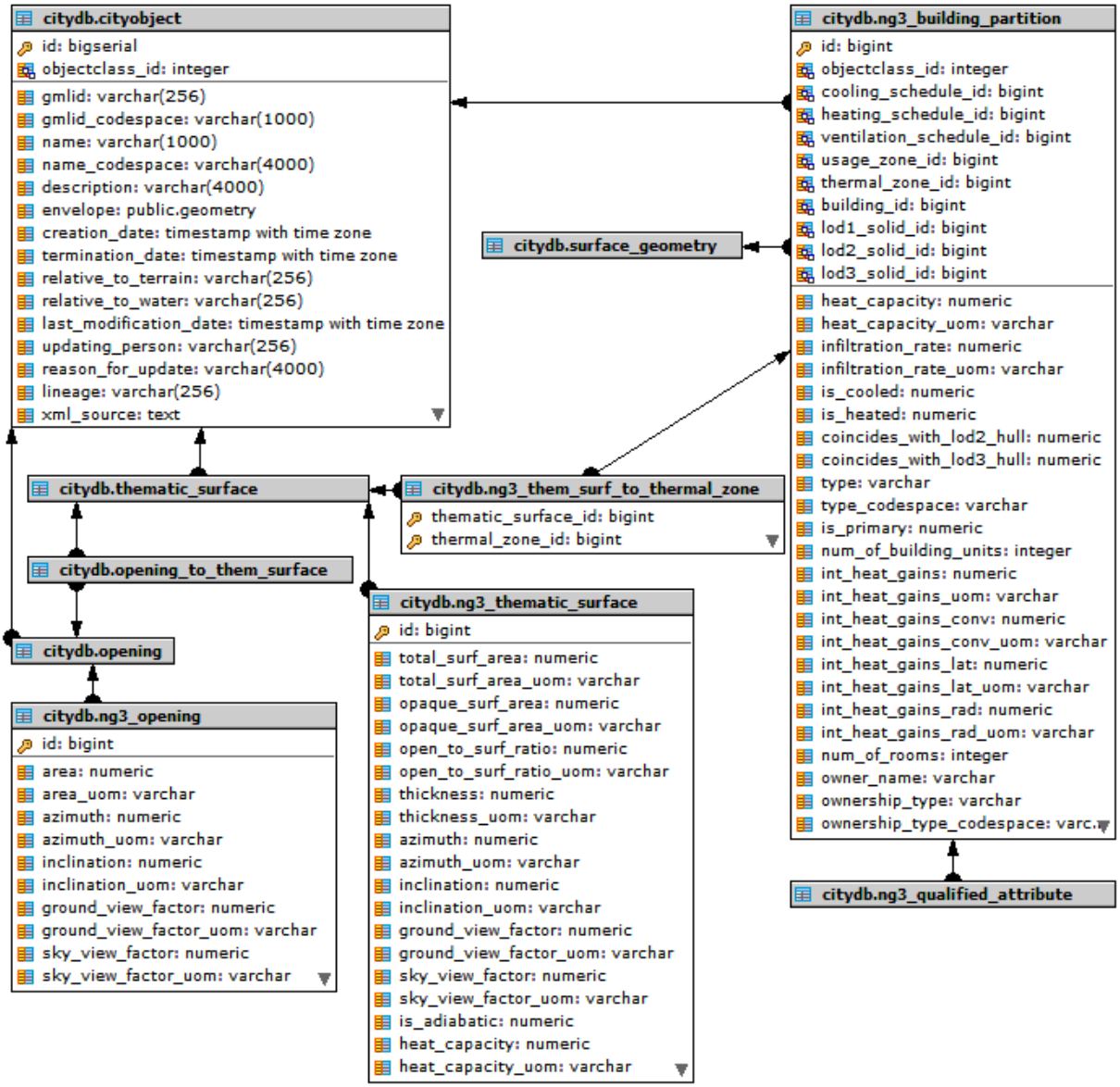


Figure 88. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Building physics module.


citydb.cityobject
id: bigserial
objectclass_id: integer
gmlid: varchar(256)
gmlid_codespace: varchar(1000)
name: varchar(1000)
name_codespace: varchar(4000)
description: varchar(4000)
envelope: public.geometry
creation_date: timestamp with time zone
termination_date: timestamp with time zone
relative_to_terrain: varchar(256)
relative_to_water: varchar(256)
last_modification_date: timestamp with time zone
updating_person: varchar(256)
reason_for_update: varchar(4000)
lineage: varchar(256)
xml_source: text
citydb.ng3_building_partition
id: bigint
objectclass_id: integer
cooling_schedule_id: bigint
heating_schedule_id: bigint
ventilation_schedule_id: bigint
usage_zone_id: bigint
thermal_zone_id: bigint
building_id: bigint
lod1_solid_id: bigint
lod2_solid_id: bigint
lod3_solid_id: bigint
heat_capacity: numeric
heat_capacity_uom: varchar
infiltration_rate: numeric
infiltration_rate_uom: varchar
is_cooled: numeric
is_heated: numeric
coincides_with_lod2_hull: numeric
coincides_with_lod3_hull: numeric
type: varchar
type_codespace: varchar
is_primary: numeric
num_of_building_units: integer
int_heat_gains: numeric
int_heat_gains_uom: varchar
int_heat_gains_conv: numeric
int_heat_gains_conv_uom: varchar
int_heat_gains_lat: numeric
int_heat_gains_lat_uom: varchar
int_heat_gains_rad: numeric
int_heat_gains_rad_uom: varchar
num_of_rooms: integer
owner_name: varchar
ownership_type: varchar
ownership_type_codespace: varc...
citydb.surface_geometry
citydb.thematic_surface
citydb.ng3_them_surf_to_thermal_zone
thematic_surface_id: bigint
thermal_zone_id: bigint
citydb.opening_to_them_surface
citydb.ng3_thematic_surface
id: bigint
total_surf_area: numeric
total_surf_area_uom: varchar
opaque_surf_area: numeric
opaque_surf_area_uom: varchar
open_to_surf_ratio: numeric
open_to_surf_ratio_uom: varchar
thickness: numeric
thickness_uom: varchar
azimuth: numeric
azimuth_uom: varchar
inclination: numeric
inclination_uom: varchar
ground_view_factor: numeric
ground_view_factor_uom: varchar
sky_view_factor: numeric
sky_view_factor_uom: varchar
is_adiabatic: numeric
heat_capacity: numeric
heat_capacity_uom: varchar
citydb.opening
citydb.ng3_opening
id: bigint
area: numeric
area_uom: varchar
azimuth: numeric
azimuth_uom: varchar
inclination: numeric
inclination_uom: varchar
ground_view_factor: numeric
ground_view_factor_uom: varchar
sky_view_factor: numeric
sky_view_factor_uom: varchar
citydb.ng3_qualified_attribute

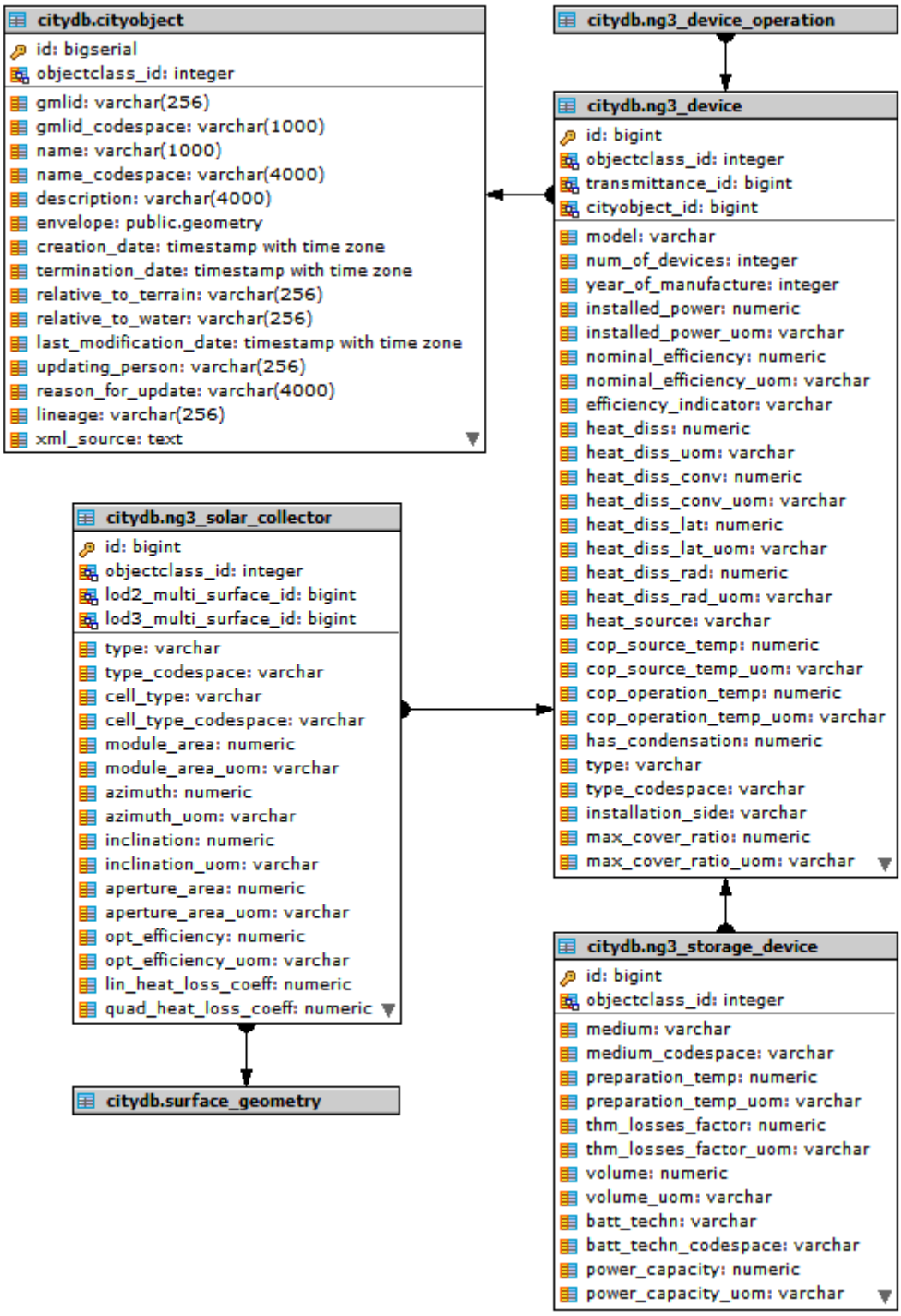


Figure 89. ER-model of the Energy ADE 3.0 Building physics module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 27, some *ThermalZone* data are extracted from the 3DCityDB and shown in Figure 90. The resulting table is split over multiple lines due to its width.

```
-- Extract some ThermalZone data
WITH co AS (
SELECT co.*, o.classname FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname = 'ThermalZone'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description,
co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM co JOIN citydb.ng3_building_partition AS t ON (co.id = t.id)
ORDER BY co.id LIMIT 2;
```

Listing 27. Example of SQL query to extract *ThermalZone* data from the database.

co_id bigint	objectclass_id integer	classname character varyin	gmlid character varying (25	gmlid_codespace character varying (	name character varying	name_codespace character varying	description character varying (4000)	envelope geometry	creation_date timestamp with	termination_date timestamp with ti
738	11081	ThermalZone	id_thermal_zone_1	[null]	ThermalZone 1	[null]	This is ThermalZone 1	01030000A040710...	[null]	[null]
739	11081	ThermalZone	id_thermal_zone_2	[null]	ThermalZone 2	[null]	This is ThermalZone 2	01030000A040710...	[null]	[null]

relative_to_terrain character varying (	relative_to_water character varying (	id bigint	objectclass_id integer	heat_capacity numeric	heat_capacity_uom character varying	infiltration_rate numeric	infiltration_rate_uom character varying	is_cooled numeric	is_heated numeric	coincides_with_lod2_hull numeric	coincides_with_lod3_hull numeric
[null]	[null]	738	11081	500	J/K	0.3	1/h	0	1	0	0
[null]	[null]	739	11081	500	J/K	0.3	1/h	0	1	0	0

type charact	type_co charact	is_prima numeric	num_of. integer	int_heat numeric	int_heat charact	int_heat numeric	int_heat charact	int_heat numeric	int_heat charact	int_heat numeric	int_heat charact	num_of. integer	owner_r charact	ownersh charact	ownersh charact	cooling. bigint	heating. bigint	ventilati bigint	usage_z bigint	thermal bigint
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]

Unused columns. Required only for *UsageZone* and *BuildingUnit* data

building_id bigint	lod1_solid_id bigint	lod2_solid_id bigint	lod3_solid_id bigint
184	1764	1772	1781
207	1790	1798	1807

Figure 90. Example of *ThermalZone* data in the 3DCityDB.

Using the SQL query in Listing 28, some building thematic surface data are extracted from the 3DCityDB and shown in Figure 91. The resulting table is split over multiple lines due to its width.

```
-- Extract some building thematic surface data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) JOIN citydb.objectclass AS o2 ON (o.superclass_id = o2.id)
WHERE o2.classname = '_BuildingBoundarySurface'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM co LEFT JOIN citydb.ng3_thematic_surface AS t ON (co.id = t.id)
ORDER BY co.classname;
```

Listing 28. Example of SQL query to extract building thematic surface data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (	name character varying (1000)	name_codespace character varying
608	11083	BasementCeilingSurface	id_thermal_zone_11_1_thermal_boundary_4	[null]	ThermalZone 11_1 ThermalBoundary 4	[null]
526	35	BuildingGroundSurface	id_thermal_zone_1_groundsurface_1	[null]	(ThermalBoundary) GroundSurface 1 (T...	[null]
528	33	BuildingRoofSurface	id_thermal_zone_1_roofsurface_1	[null]	(ThermalBoundary) RoofSurface 1 (The...	[null]
530	34	BuildingWallSurface	id_thermal_zone_1_wallsurface_1	[null]	(ThermalBoundary) WallSurface 1 (Ther...	[null]
615	11084	IntermediateFloorSurface	id_thermal_zone_11_2_thermal_boundary_5	[null]	ThermalZone_11_2 ThermalBoundary 5	[null]
527	11085	PartyWallSurface	id_thermal_zone_1_partywallsurface_1	[null]	(ThermalBoundary) PartyWallSurface 8	[null]

description character varying (4000)	envelope geometry	creation_date timestamp with	termination_date timestamp with tir	relative_to_terrain character varying (	relative_to_water character varying	id bigint	total_surf_area numeric
This is ThermalZone 11_1 ThermalBoundary 4	01030000A0407...	[null]	[null]	[null]	[null]	608	100
This is (ThermalBoundary) GroundSurface 1 (T...	01030000A0407...	[null]	[null]	[null]	[null]	526	100
This is (ThermalBoundary) RoofSurface 1 (Ther...	01030000A0407...	[null]	[null]	[null]	[null]	528	70.7107
This is (ThermalBoundary) WallSurface 1 (Ther...	01030000A0407...	[null]	[null]	[null]	[null]	530	125
This is ThermalZone_11_2 ThermalBoundary 5	01030000A0407...	[null]	[null]	[null]	[null]	615	100
This is (ThermalBoundary) PartyWallSurface 8	01030000A0407...	[null]	[null]	[null]	[null]	527	100

total_surf_area_uom character varying	opaque_surf_area numeric	opaque_surf_area_uom character varying	open_to_surf_ratio numeric	open_to_surf_ratio_uom character varying	thickness numeric	thickness_uom character varyin	azimuth numeric	azimuth_uom character varying
m^2	25	m^2	0.25	unit interval	0.5	m	-1	decimal degree
m^2	[null]	[null]	0	unit interval	[null]	[null]	-1	decimal degree
m^2	[null]	[null]	0.25	unit interval	[null]	[null]	270	decimal degree
m^2	97.9706	m^2	0.22	unit interval	[null]	[null]	180	decimal degree
m^2	0	m^2	0	unit interval	0.5	m	-1	decimal degree
m^2	[null]	[null]	0	unit interval	[null]	[null]	90	decimal degree

ground_view_factor numeric	ground_view_factor_uom character varying	sky_view_factor numeric	sky_view_factor_uom character varying	is_adiabatic numeric	heat_capacity numeric	heat_capacity_uom character varying
0.4	unit interval	0.6	unit interval	0	[null]	[null]
1	unit interval	0	unit interval	0	[null]	[null]
0.5	unit interval	0.6	unit interval	0	[null]	[null]
0.5	unit interval	0.6	unit interval	0	[null]	[null]
0	unit interval	0	unit interval	1	[null]	[null]
0	unit interval	0	unit interval	0	[null]	[null]

Figure 91. Example of building thematic surface data in the 3DCityDB.

Finally, using the SQL query in Listing 29, some building opening data are extracted from the 3DCityDB and shown in Figure 92. The resulting table is split over multiple lines due to its width.

```
-- Extract some building opening data
```

```
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) JOIN citydb.objectclass AS o2 ON (o.superclass_id = o2.id)
WHERE o2.classname = '_BuildingOpening'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM co LEFT JOIN citydb.ng3_opening AS t ON (co.id = t.id)
ORDER BY co.classname, co.id LIMIT 2;
```

Listing 29. Example of SQL query to extract building opening data from the database.

co_id bigint	objectclass_id integer	classname character varying	gmlid character varying (256)	gmlid_codespace character varying	name character varying (10	name_codespace character varying
264	38	BuildingWindow	id_thermal_zone_1_wallsurface_1_thermal_opening_1	[null]	Thermal Opening 1	[null]
265	38	BuildingWindow	id_thermal_zone_1_wallsurface_2_thermal_opening_2	[null]	Thermal Opening 2	[null]

description character varying (4000)	envelope geometry	creation_date timestamp witl	termination_date timestamp with ti	relative_to_terrain character varying	relative_to_water character varying	id bigint	area numeric	area_uom character v	azimuth numeric
This is Thermal Opening 1	01030000A04071...	[null]	[null]	[null]	[null]	264	27.0294	m^2	180
This is Thermal Opening 2	01030000A04071...	[null]	[null]	[null]	[null]	265	27.0294	m^2	0

azimuth_uom character varying	inclination numeric	inclination_uom character varying	ground_view_factor numeric	ground_view_factor_uom character varying	sky_view_factor numeric	sky_view_factor_uom character varying
decimal degree	90	decimal degree	0.5	unit interval	0.6	unit interval
decimal degree	90	decimal degree	0.5	unit interval	0.6	unit interval

Figure 92. Example of building opening data in the 3DCityDB.

7.7.10 Devices module

In the Devices module, the mapping rules described in section 7.6.3 are applied. Class *AbstractDevice* and some other classes (e.g. *HeatPump*, *Boiler*, etc.) – please refer to Figure 93 – are mapped to one unique target table NG3_DEVICE. All properties up to class *_CityObject* are written to table CITYOBJECT, then all properties of class *AbstractDevice* (and some of its subclasses) are written to table NG3_DEVICE. This means that, depending on the specific class, some columns will be written, while others may remain empty. The relation between ADE *_CityObject* and *AbstractDevice* is realised by means of the foreign key in column *cityobject_id*.
The *transmittance* property of class *MovableShadingDevice* is written to table NG3_OPTICAL_PROPERTY and referenced by means of a foreign key in column *transmittance_id*.
Class *AbstractSolarCollector* and all its subclasses are mapped to the linked table NG3_SOLAR_COLLECTOR. The *id* column of this table acts as primary key and, at the same time, foreign key to table NG3_DEVICE. The multisurface geometries of class *AbstractSolarCollector* are written to table SURFACE_GEOMETRY following the usual 3DCityDB mapping rules for aggregate geometries.
Likewise, classes *ThermalStorageDevice* and *ElectricalStorageDevice* are mapped to one joined table NG3_STORAGE_DEVICE. The *id* column of this table acts as primary key and, at the same time, foreign key to table NG3_DEVICE.
Figure 93 represents graphically the OO-to-ER-model mapping of the Devices module while Figure 94 depicts the resulting ER-model.

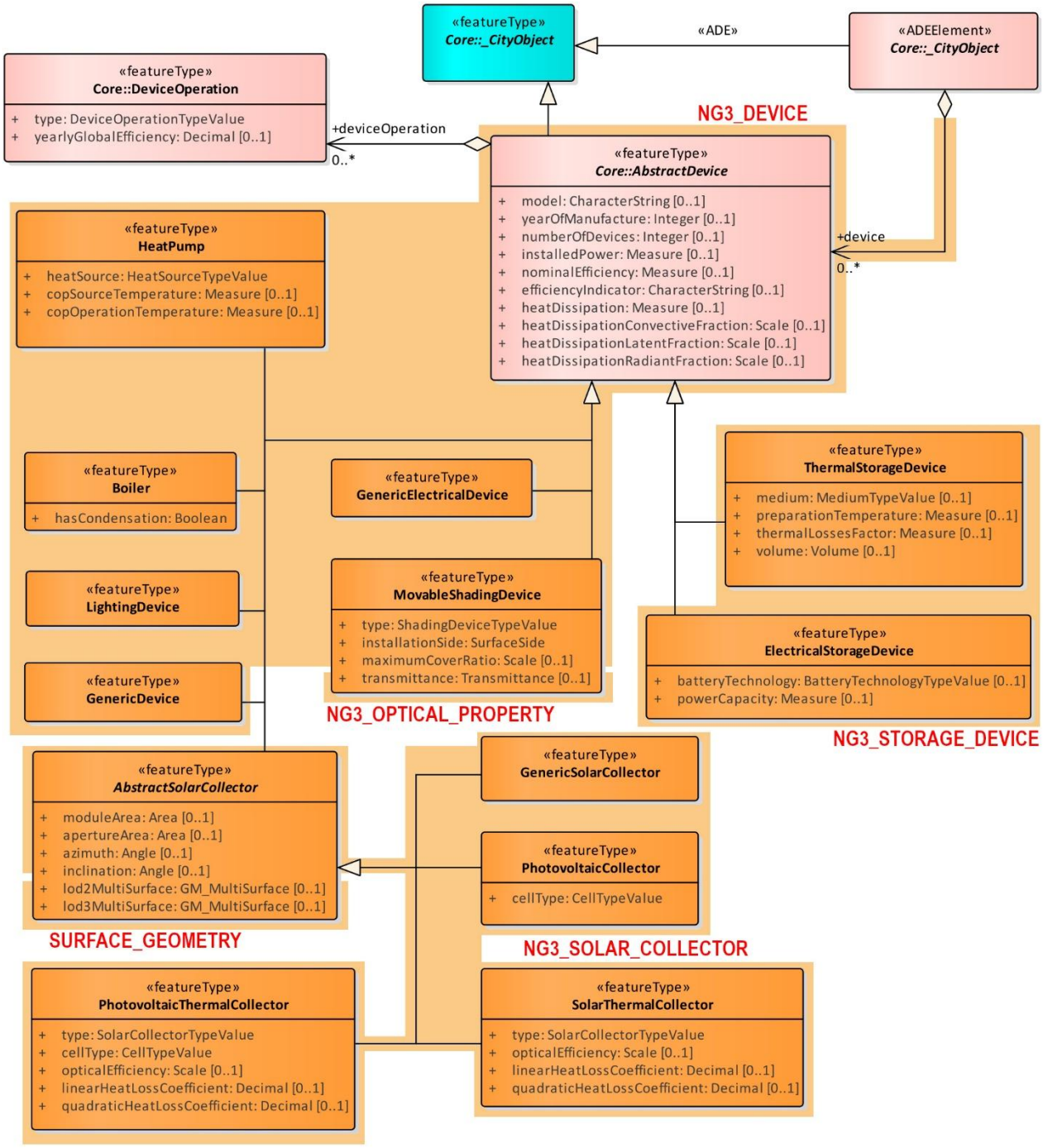
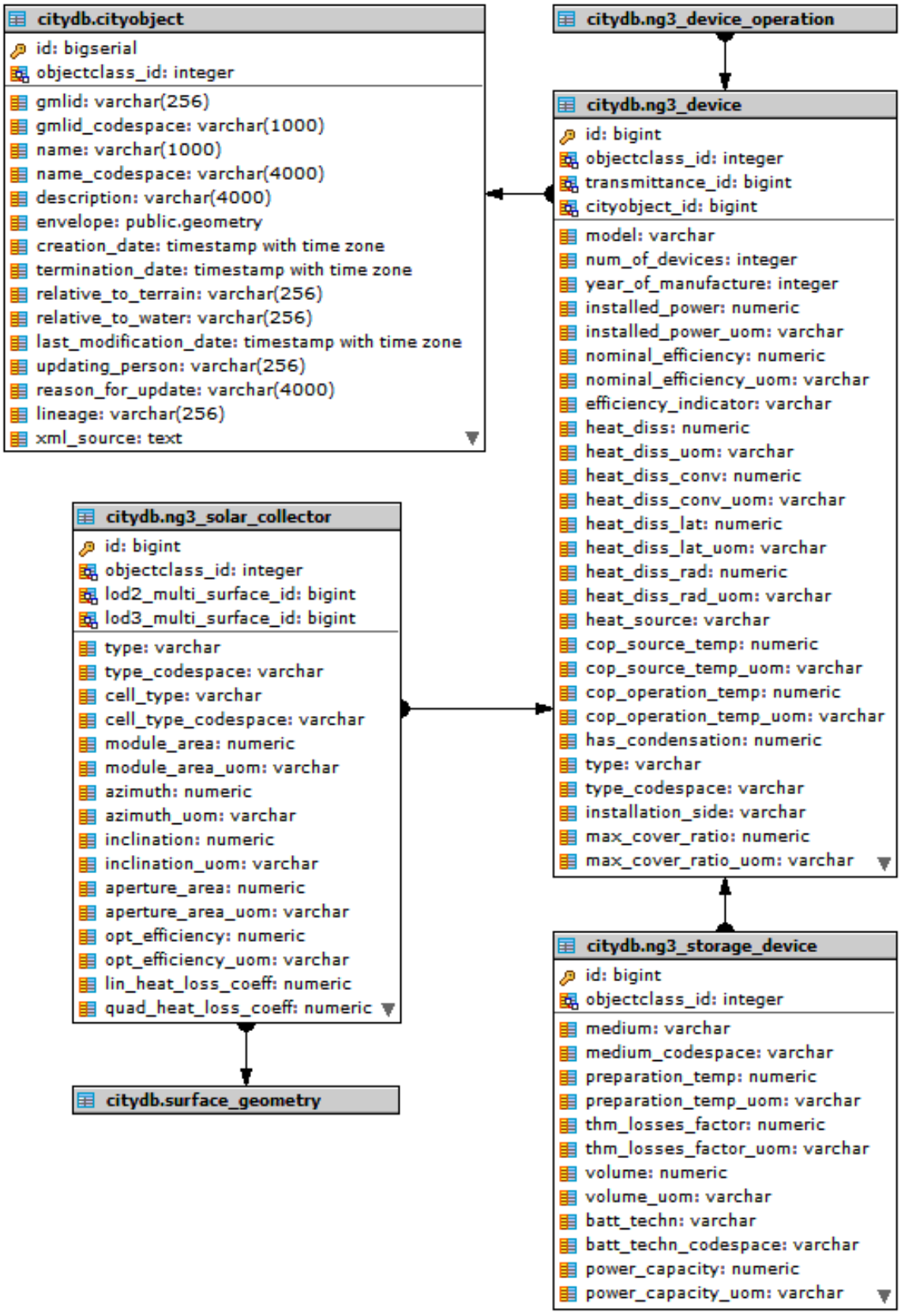


Figure 93. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Devices module.


citydb.cityobject
id: bigserial
objectclass_id: integer
gmlid: varchar(256)
gmlid_codespace: varchar(1000)
name: varchar(1000)
name_codespace: varchar(4000)
description: varchar(4000)
envelope: public.geometry
creation_date: timestamp with time zone
termination_date: timestamp with time zone
relative_to_terrain: varchar(256)
relative_to_water: varchar(256)
last_modification_date: timestamp with time zone
updating_person: varchar(256)
reason_for_update: varchar(4000)
lineage: varchar(256)
xml_source: text
citydb.ng3_device_operation
citydb.ng3_device
id: bigint
objectclass_id: integer
transmittance_id: bigint
cityobject_id: bigint
model: varchar
num_of_devices: integer
year_of_manufacture: integer
installed_power: numeric
installed_power_uom: varchar
nominal_efficiency: numeric
nominal_efficiency_uom: varchar
efficiency_indicator: varchar
heat_diss: numeric
heat_diss_uom: varchar
heat_diss_conv: numeric
heat_diss_conv_uom: varchar
heat_diss_lat: numeric
heat_diss_lat_uom: varchar
heat_diss_rad: numeric
heat_diss_rad_uom: varchar
heat_source: varchar
cop_source_temp: numeric
cop_source_temp_uom: varchar
cop_operation_temp: numeric
cop_operation_temp_uom: varchar
has_condensation: numeric
type: varchar
type_codespace: varchar
installation_side: varchar
max_cover_ratio: numeric
max_cover_ratio_uom: varchar
citydb.ng3_solar_collector
id: bigint
objectclass_id: integer
lod2_multi_surface_id: bigint
lod3_multi_surface_id: bigint
type: varchar
type_codespace: varchar
cell_type: varchar
cell_type_codespace: varchar
module_area: numeric
module_area_uom: varchar
azimuth: numeric
azimuth_uom: varchar
inclination: numeric
inclination_uom: varchar
aperture_area: numeric
aperture_area_uom: varchar
opt_efficiency: numeric
opt_efficiency_uom: varchar
lin_heat_loss_coeff: numeric
quad_heat_loss_coeff: numeric
citydb.ng3_storage_device
id: bigint
objectclass_id: integer
medium: varchar
medium_codespace: varchar
preparation_temp: numeric
preparation_temp_uom: varchar
thm_losses_factor: numeric
thm_losses_factor_uom: varchar
volume: numeric
volume_uom: varchar
batt_techn: varchar
batt_techn_codespace: varchar
power_capacity: numeric
power_capacity_uom: varchar
citydb.surface_geometry

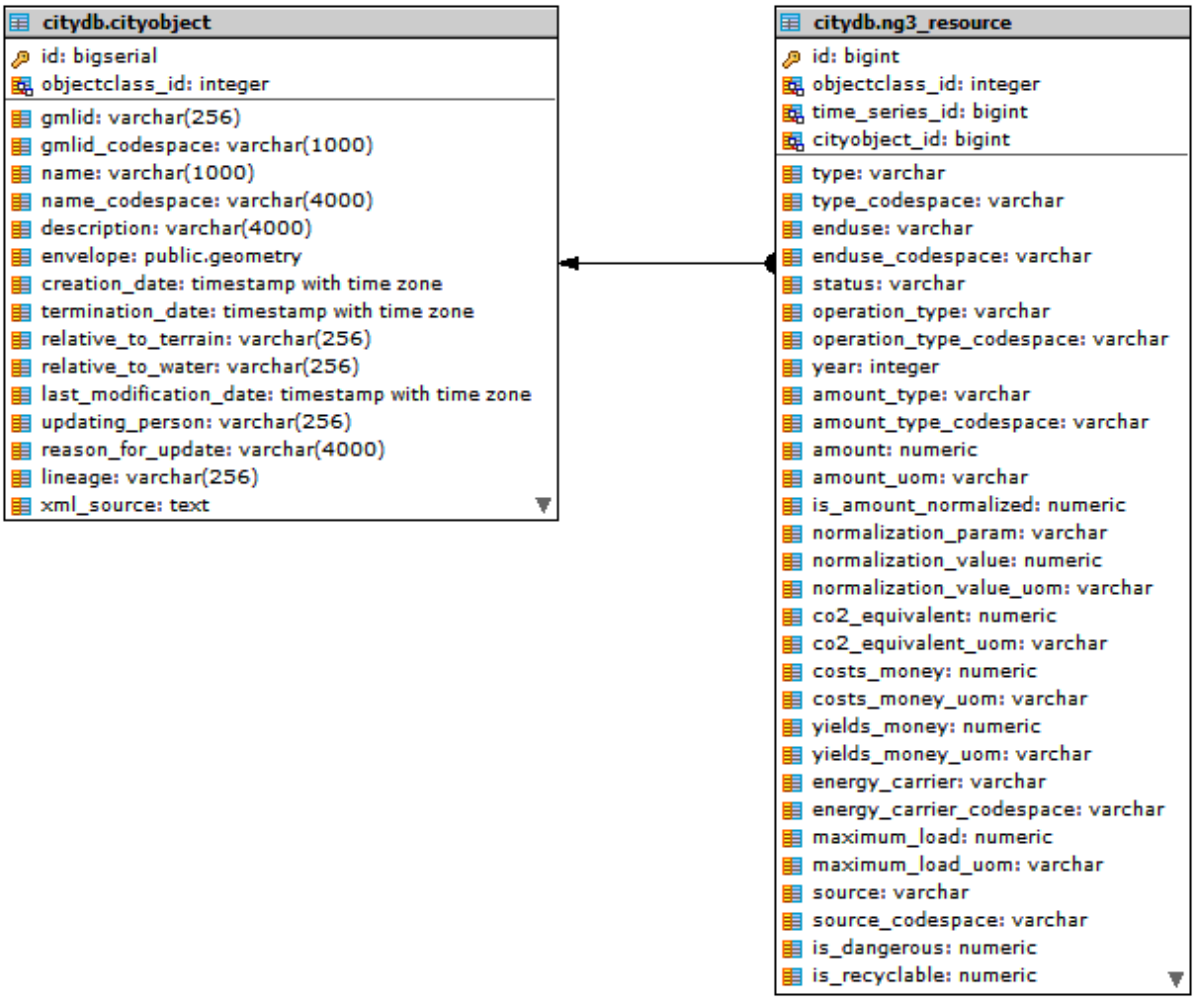


Figure 94. ER-model of the Energy ADE 3.0 Device module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 30, some device data are extracted from the 3DCityDB and shown in Figure 95. The resulting table is split over multiple lines due to its width.

```
-- Extract some device data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) JOIN citydb.objectclass AS o2 ON (o.superclass_id = o2.id)
WHERE o2.classname = 'AbstractDevice'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*
FROM co LEFT JOIN citydb.ng3_device AS t ON (co.id = t.id)
ORDER BY co.classname;
```

Listing 30. Example of SQL query to extract *ThermalZone* data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (…	name character varying (1000)	name_codespace character varying …
480	11102	Boiler	id_boiler_4	[null]	Boiler 4	[null]
733	11108	ElectricalStorageDevice	id_electrical_storage_device_11	[null]	ElectricalStorageDevice 11	[null]
483	11103	GenericDevice	id_generic_device_1	[null]	GenericDevice 1	[null]
487	11104	GenericElectricalDevice	id_generic_electrical_device_2	[null]	GenericElectricalDevice 2	[null]
490	11106	HeatPump	id_heat_pump_5	[null]	HeatPump 5	[null]
493	11105	LightingDevice	id_lighting_device_3	[null]	LightingDevice 3	[null]
497	11107	MovableShadingDevice	id_movable_shading_device_6	[null]	MovableShadingDevice 6	[null]
736	11109	ThermalStorageDevice	id_thermal_storage_device_7	[null]	ThermalStorageDevice 7	[null]

description character varying (4000)	envelope geometry	creation_date timestamp with…	termination_date timestamp with ti…	relative_to_terrain character varying (…	relative_to_water character varying	id bigint	objectclass_id integer
This is Boiler 4	01030000A04…	[null]	[null]	[null]	[null]	480	11102
This is ElectricalStorageDevice 11	01030000A04…	[null]	[null]	[null]	[null]	733	11108
This is GenericDevice 1	01030000A04…	[null]	[null]	[null]	[null]	483	11103
This is GenericElectricalDevice 2	01030000A04…	[null]	[null]	[null]	[null]	487	11104
This is HeatPump 5	01030000A04…	[null]	[null]	[null]	[null]	490	11106
This is LightingDevice 3	01030000A04…	[null]	[null]	[null]	[null]	493	11105
This is MovableShadingDevice 6	01030000A04…	[null]	[null]	[null]	[null]	497	11107
This is ThermalStorageDevice 7	01030000A04…	[null]	[null]	[null]	[null]	736	11109

model character varying	num_of_devices integer	year_of_manufacture integer	installed_power numeric	installed_power_uom character varying	nominal_efficiency numeric
Boiler Boiler 4	1	1992	9.8	kW	0.2
ElectricalStorageDevice ElectricalStorageDevice 11	1	2016	2.3	kW	0.8
GenericDevice GenericDevice 1	1	2015	6.3	kW	0
GenericElectricalDevice GenericElectricalDevice 2	1	2008	4.9	kW	0.1
HeatPump HeatPump 5	1	1996	5.5	kW	0.5
LightingDevice LightingDevice 3	1	1999	3.3	kW	0.7
MovableShadingDevice MovableShadingDevice 6	1	2006	4	kW	0.7
ThermalStorageDevice ThermalStorageDevice 7	1	1990	9.3	kW	0.2

nominal_efficiency_uom character varying	efficiency_indicator character varying	heat_diss numeric	heat_diss_uom character varying	heat_diss_conv numeric	heat_diss_conv_uom character varying	heat_diss_lat numeric	heat_diss_lat_uom character varying
unit interval	eff_indic_string_4	10	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_11	17	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_1	21	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_2	14	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_5	13	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_3	1	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_6	5	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_7	27	W/m^2	0.2	unit interval	0.3	unit interval

heat_diss_rad numeric	heat_diss_rad_uom character varying	heat_source character vary…	cop_source_temp numeric	cop_source_temp_uom character varying	cop_operation_temp numeric	cop_operation_temp_uom character varying
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	aquifer	5	degrees Celsius	10	degrees Celsius
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	[null]	[null]	[null]	[null]	[null]
0.5	unit interval	[null]	[null]	[null]	[null]	[null]

has_condensation numeric	type character varying	type_codespace character varying	installation_side character varying	max_cover_ratio numeric	max_cover_ratio_uom character varying	transmittance_id bigint	cityobject_id bigint
1	[null]	[null]	[null]	[null]	[null]	[null]	207
[null]	[null]	[null]	[null]	[null]	[null]	[null]	239
0	[null]	[null]	[null]	[null]	[null]	[null]	184
0	[null]	[null]	[null]	[null]	[null]	[null]	184
0	[null]	[null]	[null]	[null]	[null]	[null]	170
0	[null]	[null]	[null]	[null]	[null]	[null]	207
0	shutter	shading_devi...	outside	0.7	unit inteval	31	170
[null]	[null]	[null]	[null]	[null]	[null]	[null]	192

Figure 95. Example of device data in the 3DCityDB.

Using the SQL query in Listing 31, some device data are extracted from the 3DCityDB and shown in Figure 96. The resulting table is split over multiple lines due to its width.

```
-- Extract some solar collector data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) JOIN citydb.objectclass AS o2 ON (o.superclass_id = o2.id)
WHERE o2.classname = 'AbstractSolarCollector'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*, t2.*
FROM co JOIN citydb.ng3_device AS t ON (co.id = t.id) JOIN citydb.ng3_solar_collector AS t2 ON (t.id = t2.id)
ORDER BY co.classname;
```

Listing 31. Example of SQL query to extract solar collector data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (*	name character varying (1000)	name_codespace character varying
659	11111	GenericSolarCollector	id_generic_solar_collector_7	[null]	GenericSolarCollector 7	[null]
669	11112	PhotovoltaicCollector	id_pv_collector_17	[null]	PhotoVoltaicCollector 17	[null]
685	11114	PhotovoltaicThermalCollector	id_pv_thermal_collector_49	[null]	PhotovoltaicThermalCollector 49	[null]
701	11113	SolarThermalCollector	id_solar_thermal_collector_33	[null]	SolarThermalCollector 33	[null]

description character varying (4000)	envelope geometry	creation_date timestamp witl	termination_date timestamp with ti	relative_to_terrain character varying (	relative_to_water character varying	id bigint	objectclass_id integer
This is GenericSolarCollector 7	01030000A040...	[null]	[null]	[null]	[null]	659	11111
This is PhotoVoltaicCollector 17	01030000A040...	[null]	[null]	[null]	[null]	669	11112
This is PhotovoltaicThermalCollector 49	01030000A040...	[null]	[null]	[null]	[null]	685	11114
This is SolarThermalCollector 33	01030000A040...	[null]	[null]	[null]	[null]	701	11113

model character varying	num_of_devices integer	year_of_manufacture integer	installed_power numeric	installed_power_uom character varying	nominal_efficiency numeric
GenericSolarCollector Model GenericSolarCollector 7	3	2015	9	kWp	0.3
PhotoVoltaicCollector Model PhotoVoltaicCollector 17	1	2017	5	kWp	0.3
PhotovoltaicThermalModel PhotovoltaicThermalCollector 49	1	2021	5	kWp	0.3
SolarThermalModel SolarThermalCollector 33	1	2019	5	kWp	0.3

efficiency_indicator character varying	heat_diss numeric	heat_diss_uom character varyin	heat_diss_conv numeric	heat_diss_conv_uom character varying	heat_diss_lat numeric	heat_diss_lat_uom character varying	heat_diss_rad numeric	heat_diss_rad_uom character varying
eff_indic_string_7	15	W/m^2	0.1	unit interval	0.5	unit interval	0.4	unit interval
eff_indic_string_...	15	W/m^2	0.1	unit interval	0.5	unit interval	0.4	unit interval
eff_indic_string_...	15	W/m^2	0.1	unit interval	0.5	unit interval	0.4	unit interval
eff_indic_string_...	15	W/m^2	0.1	unit interval	0.5	unit interval	0.4	unit interval

heat_so character	cop_sou numeric	cop_sou character	cop_ope numeric	cop_ope character	has_cor numeric	type character varying	type_codespace character varying	installat character	max_co numeric	max_co character	transmi bigint	cityobject_id bigint
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	191
[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	184
[null]	[null]	[null]	[null]	[null]	[null]	flatPlaneCollector	solar_coll_codeSpace_2	[null]	[null]	[null]	[null]	231
[null]	[null]	[null]	[null]	[null]	[null]	flatPlaneCollector	solar_coll_codeSpace_1	[null]	[null]	[null]	[null]	192

Unused columns.

Unused columns.

id bigint	objectclass_id integer	type character varying	type_codespace character varying	cell_type character varying	cell_type_codespace character varying	module_area numeric	module_area_uom character varying
659	11111	[null]	[null]	[null]	[null]	27.36	m^2
669	11112	[null]	[null]	polycristalline	cell_type_codeSpace_1	8.16	m^2
685	11114	flatPlaneCollector	solar_coll_codeSpace_2	monocristalline	cell_type_codeSpace_2	8.16	m^2
701	11113	flatPlaneCollector	solar_coll_codeSpace_1	[null]	[null]	8.16	m^2

azimuth numeric	azimuth_uom character varying	inclination numeric	inclination_uom character varying	aperture_area numeric	aperture_area_uom character varying	opt_efficiency numeric	opt_efficiency_uom character varying	lin_heat_loss_coeff numeric
180	decimal degrees	45	decimal degrees	21.888	m^2	[null]	[null]	[null]
270	decimal degrees	45	decimal degrees	7.752	m^2	[null]	[null]	[null]
90	decimal degrees	45	decimal degrees	7.752	m^2	0.8	unit interval	10
180	decimal degrees	45	decimal degrees	7.752	m^2	0.4	unit interval	10

quad_heat_loss_coeff numeric	lod2_multi_surface_id bigint	lod3_multi_surface_id bigint
[null]	3541	[null]
[null]	[null]	3561
0.68	[null]	3593
0.68	[null]	3625

Figure 96. Example of solar collector data in the 3DCityDB.

Finally, using the SQL query in Listing 32, some storage device data are extracted from the 3DCityDB and shown in Figure 97. The resulting table is split over multiple lines due to its width.

```
-- Extract some storage device data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id)
WHERE o.classname IN ('ElectricalStorageDevice', 'ThermalStorageDevice')
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace,
co.description, co.envelope, co.creation_date, co.termination_date, co.relative_to_terrain, co.relative_to_water, t.*, t2.*
FROM co JOIN citydb.ng3_device AS t ON (co.id = t.id) JOIN citydb.ng3_storage_device AS t2 ON (t.id = t2.id)
ORDER BY co.classname;
```

Listing 32. Example of SQL query to extract storage device data from the database.

co_id bigint	objectclass_id integer	classname character varying (256)	gmlid character varying (256)	gmlid_codespace character varying (1	name character varying (1000)	name_codespace character varying
733	11108	ElectricalStorageDevice	id_electrical_storage_device_11	[null]	ElectricalStorageDevice 11	[null]
736	11109	ThermalStorageDevice	id_thermal_storage_device_7	[null]	ThermalStorageDevice 7	[null]

description character varying (4000)	envelope geometry	creation_date timestamp wit	termination_date timestamp with ti	relative_to_terrain character varying (	relative_to_water character varying	id bigint	objectclass_id integer
This is ElectricalStorageDevice 11	01030000A0407...	[null]	[null]	[null]	[null]	733	11108
This is ThermalStorageDevice 7	01030000A0407...	[null]	[null]	[null]	[null]	736	11109

model character varying	num_of_devices integer	year_of_manufacture integer	installed_power numeric	installed_power_uom character varying	nominal_efficiency numeric
ElectricalStorageDevice ElectricalStorageDevice 11	1	2016	2.3	kW	0.8
ThermalStorageDevice ThermalStorageDevice 7	1	1990	9.3	kW	0.2

nominal_efficiency_uom character varying	efficiency_indicator character varying	heat_diss numeric	heat_diss_uom character varying	heat_diss_conv numeric	heat_diss_conv_uom character varying	heat_diss_lat numeric	heat_diss_lat_uom character varying
unit interval	eff_indic_string_11	17	W/m^2	0.2	unit interval	0.3	unit interval
unit interval	eff_indic_string_7	27	W/m^2	0.2	unit interval	0.3	unit interval

heat_diss_rad numeric	heat_diss_rad_uom character varying	heat_so charact	cop_sou numeric	cop_sou charact	cop_ope numeric	cop_ope charact	has_cor numeric	type charact	type_co charact	installat charact	max_co numeric	max_co charact	transmi bigint	cityobject_id bigint
0.5	unit interval	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	239
0.5	unit interval	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	192

Unused columns.

id bigint	objectclass_id integer	medium character	medium_codespace character varying	preparation_temp numeric	preparation_temp_uom character varying	thm_losses_factor numeric	thm_losses_factor_uom character varying	volume numeric
733	11108	[null]	[null]	[null]	[null]	[null]	[null]	[null]
736	11109	steam	medium_codeSpace	80	degrees Celsius	80	units no idea...	50

volume_uom character vary	batt_techn character va	batt_techn_codespace character varying	power_capacity numeric	power_capacity_uom character varying
[null]	lithium	batt_tech_codeSpace	600	VAh
m^3	[null]	[null]	[null]	[null]

Figure 97. Example of storage device data in the 3DCityDB.

7.7.11 Resources module

Following the mapping rules described in section 7.6.4, and as depicted in Figure 98, all classes of the Resources module are grouped together and mapped to one single target table: NG3_RESOURCE. All attributes up to class *_Feature* are written to table CITYOBJECT, and then all attributes from class *AbstractResource* are written in table NG3_RESOURCE. This means that, depending on the specific class, some columns will be written, while others may remain empty. Relation *resource* between ADE *_CityObject* and *AbstractResource* is realised by means of the foreign key stored in column *cityobject_id*. The optional relation to *AbstractTimeSeries* is realised by means of the foreign key in column *time_series_id*.
Figure 99 depicts the resulting ER-model, while some examples are provided in Figure 100.

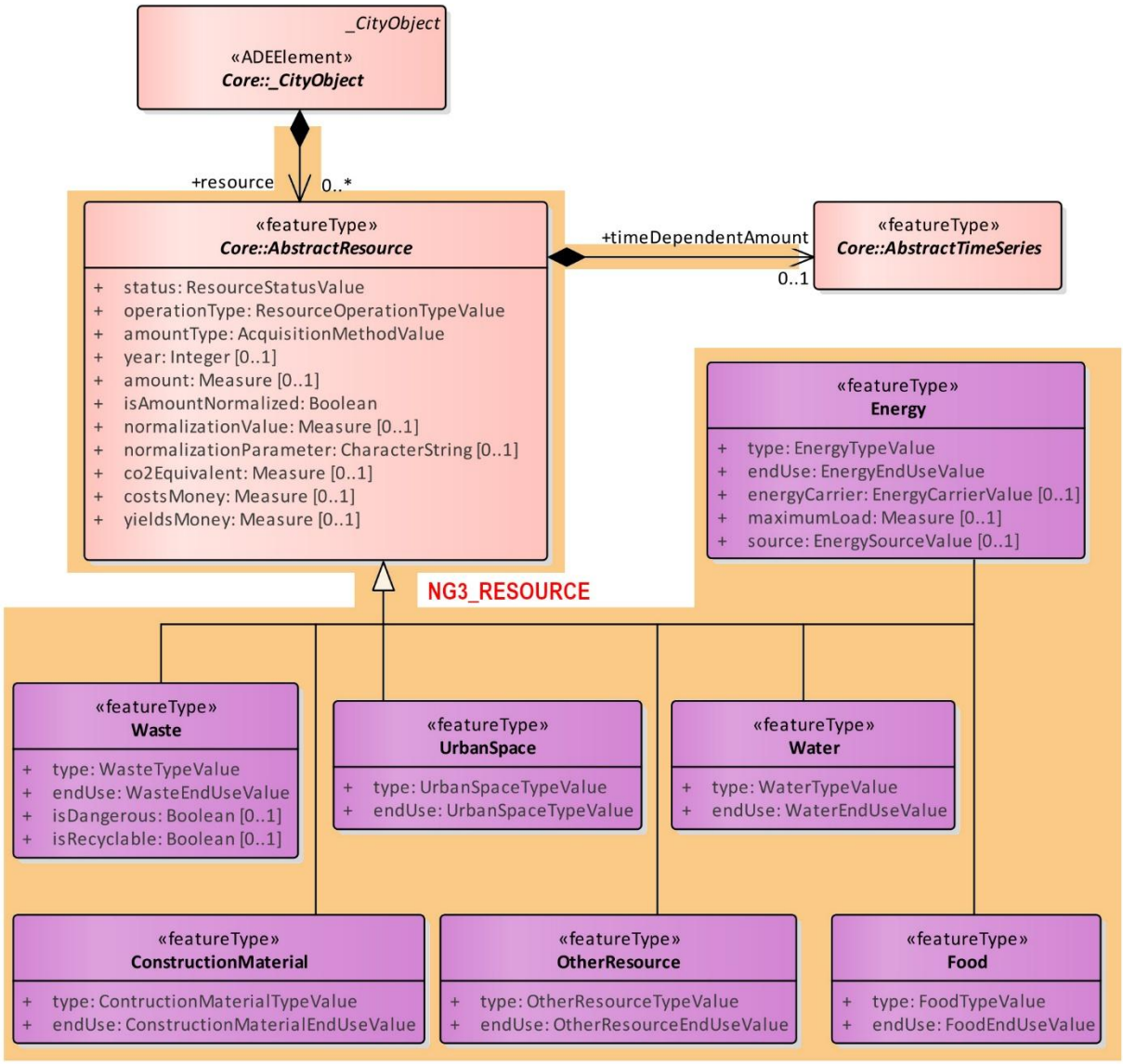


Figure 98. Graphical representation of the OO-to-ER-model mapping of the Energy ADE 3.0 Resources module.

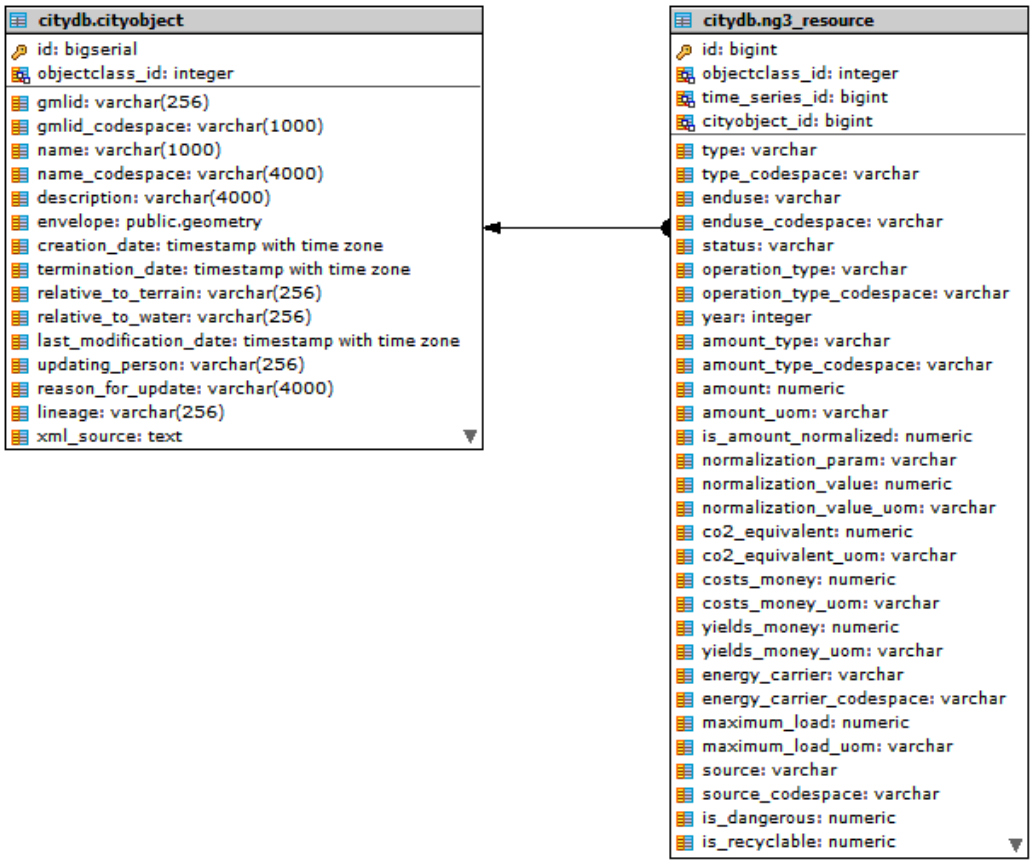


Figure 99. ER-model of the Energy ADE 3.0 Resources module.

Using the data of dataset **Alderaan_Energy_ADE_All.gml** and the SQL query in Listing 33, some resource are extracted from the 3DCityDB and shown in Figure 100. The resulting table is split over multiple lines due to its width.

```
-- Extract some resource data
WITH co AS (
SELECT DISTINCT ON (co.objectclass_id) co.*, o.classname
FROM citydb.cityobject AS co JOIN citydb.objectclass AS o ON (co.objectclass_id = o.id) JOIN citydb.objectclass AS o2 ON (o.superclass_id = o2.id)
WHERE o2.classname = 'AbstractResource'
)
SELECT co.id AS co_id, co.objectclass_id, co.classname, co.gmlid, co.gmlid_codespace, co.name, co.name_codespace, co.description, t.* FROM co LEFT JOIN citydb.ng3_resource AS t ON (co.id = t.id)
ORDER BY co.classname;
```

Listing 33. Example of SQL query to extract resource data from the database.

co_id bigint	objectclass_id integer	classname character va	gmlid character varying (2	gmlid_codespace character varying (	name character v	name_codespace character varying (	description character varying	id bigint	objectclass_id integer
856	11072	Energy	id_res_energy_1	[null]	Energy 1	[null]	This is Energy 1	856	11072
858	11077	Water	id_res_water_1	[null]	Water 1	[null]	This is Water 1	858	11077

type character var	type_codespace character varying	enduse character varying	enduse_codespace character varying	status character	operation_type character varying	operation_type_codespace character varying	year integer
primary	eng_type_codespace_1	electricalAppliances	nrg_enduse_codespace_1	actual	demands	resource-oper_codespace_1	2023
drinkWater	wat_type_codespace_1	watering	wat_enduse_codespace_1	actual	demands	resource-oper_codespace_7	2023

amount_type character varying	amount_type_codespace character varying	amount numeric	amount_uom character varyi	time_series_id bigint	is_amount_normalized numeric	normalization_param character varying	normalization_value numeric
measurement	amnt_type_codespace_1	100	kWh/a	765	0	[null]	[null]
simulation	amnt_type_codespace_7	15	m3/a	[null]	0	[null]	[null]

normalization_value_uom character varying	co2_equivalent numeric	co2_equivalent_uom character varying	costs_money numeric	costs_money_uom character varying	yields_money numeric	yields_money_uom character varying	energy_carrier character varyin
[null]	12	kg/a	50	Euro	0	Euro	electricity
[null]	[null]	[null]	15	Euro	0	Euro	[null]

energy_carrier_codespace character varying	maximum_load numeric	maximum_load_uom character varying	source character var	source_codespace character varying	is_dangerous numeric	is_recyclable numeric	cityobject_id bigint
nrg_carrier_codespace_1	4	kW	powerGrid	nrg_source_codespace_1	0	0	886
[null]	[null]	[null]	[null]	[null]	0	0	7

Figure 100. Example of resource data in the 3DCityDB.

8 FME workbench

8.1 Overview and prerequisites

In order to test the database implementation of the Energy ADE 3.0 while the Java-based libraries required by the 3D City Database Importer Exporter are still in development, a workbench for FME Form has been created as an alternative solution to import Energy ADE 3.0 data into the 3D City Database.
The workbench has been created with FME Form 2024.2.4.0 and allows to import Energy ADE 3.0 data into the chosen 3DCityDB schema. The default schema is the "citydb" one, however the user can define a different one.
It is important to highlight that, upon *each* run of the workbench, following operations will be carried out:

1) Any existing Energy ADE 3.0 data in the selected 3DCityDB schema will be deleted. Sequences will be updated accordingly
2) If applicable, the status of the “is_xlink” attribute in the SURFACE_GEOMETRY table will be updated as a result of the deleted Energy ADE 3.0 geometry data
3) New Energy ADE 3.0 will be read from the input file and written to the database. Existing objects will be updated, if necessary (e.g. Buildings), and new Energy ADE data will be inserted. This applies to feature attributes and relations, but also geometries
4) If applicable, the status of the “is_xlink” attribute in the SURFACE_GEOMETRY table will be updated again as a result of the inserted Energy ADE 3.0 geometry data
5) The 3D bounding box of the newly inserted data is computed and the *envelope* column of table CITYOBJECT is updated
6) At the end of the import process, the user will be notified with a message in the FME Data Inspector.

There are **two important prerequisites** to the current workbench. Namely:

- ***Before*** **running the FME workbench, the user must install and set up the database schema of the Energy ADE 3.0 using the ADE Manager of the 3DCityDB Importer/Exporter**.

This operation can be carried out, as mentioned in section 7.1

- ***Before*** **running the FME workbench, the user must import the “standard” CityGML data of the input file using the 3DCityDB Importer/Exporter.**

This means that no “standard” CityGML data will be imported by the FME workbench. The FME workbench imports *only* Energy ADE 3.0 data. If the file is valid, then the error messages raised by the Importer upon import can be ignored, as they refer to Energy ADE 3.0 contents that, obviously, are not recognised at the moment.

8.2 Workbench setup

In order to set up the workbench, few steps are required. After opening the workbench, the user must:

1) Define the **connection parameters** to the database
2) Set the **name of the input file** containing CityGML + Energy ADE 3.0 data

3) Set the **path to the XSD file** of the Energy ADE 3.0 (default is to the ./xsd subfolder)
4) Set the **name of the target 3DCityDB schema** (default: “citydb”).

A screenshot taken from FME is presented in Figure 101 and shows where the User parameters and the Database connection entries are in the Navigator window.

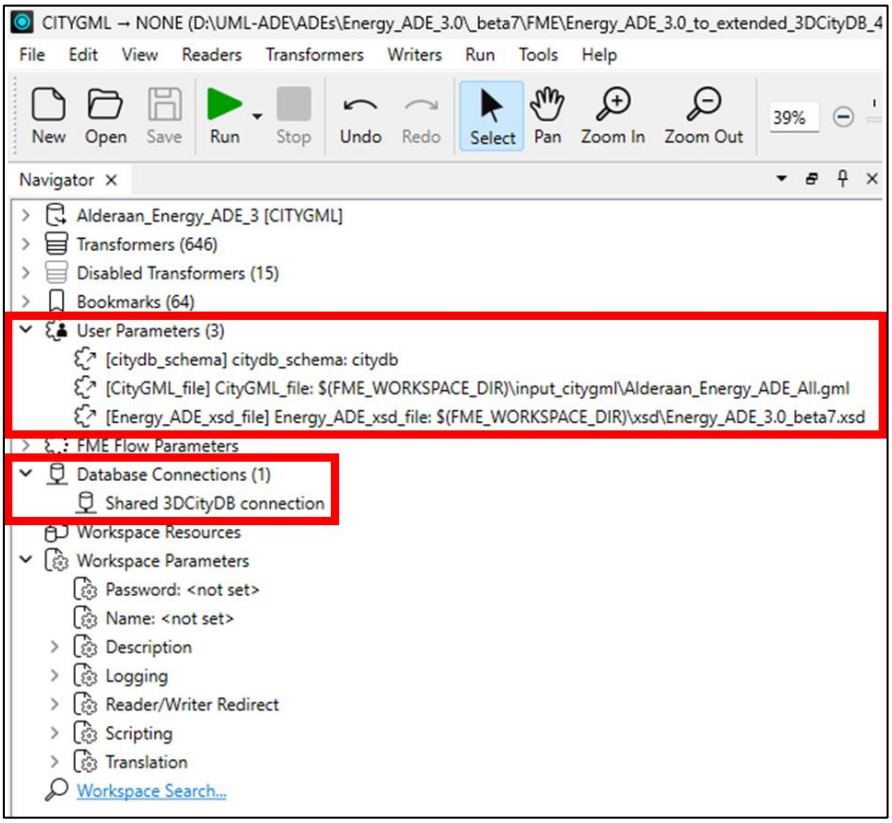
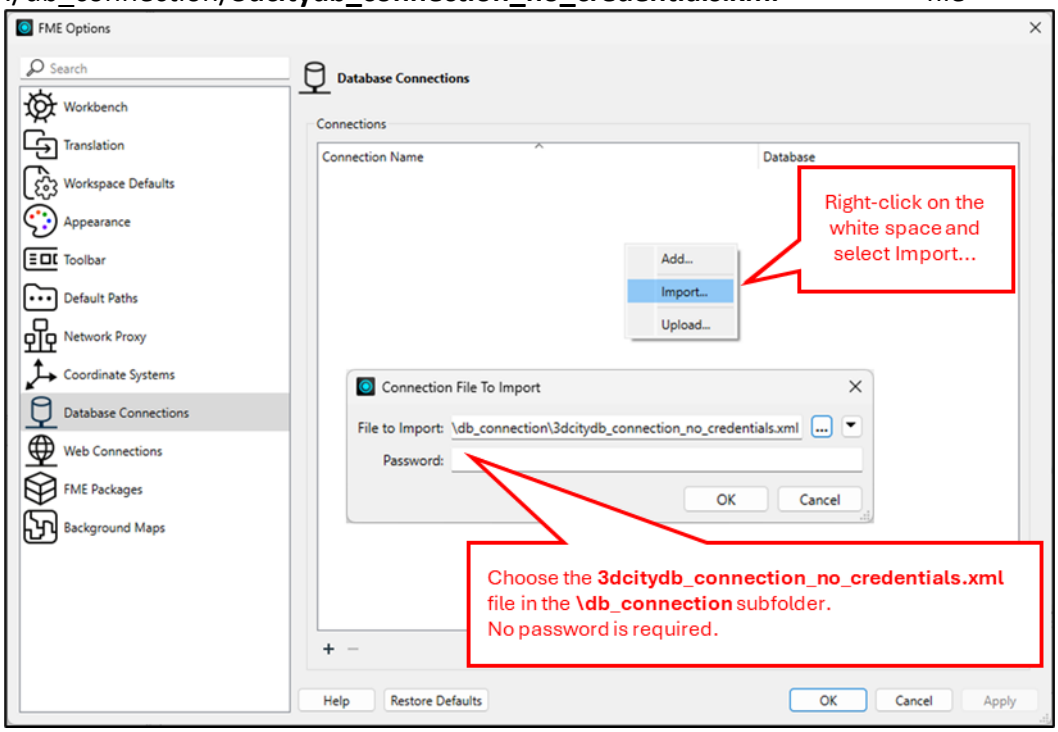


Figure 101. Example of how to set the User parameters and the Database connection parameters in FME Form.

The first time the workbench is loaded, and *before* running it, the user must add the connection and edit the connection parameters. This means that the user must:

1) Go to the menu Tools\FME Options, and choose the "Database Connections" tab

2) **Right-click in the Connections white space** and import the connection stored in the ./db_connection/**3dcitydb_connection_no_credentials.xml** file (see



3) Figure 102). A connection named "Shared 3DCityDB connection" will be loaded and appear in the Navigator tab.
4) Open it and **enter the connection parameters**, test and save the connection (see Figure 103).

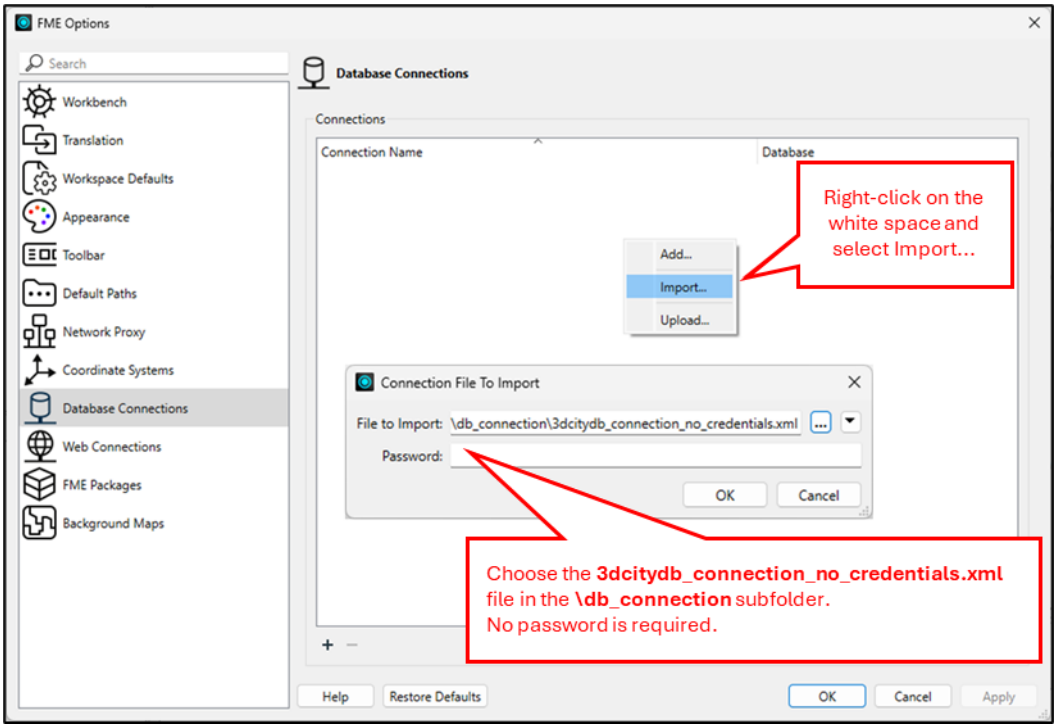


Figure 102. Example of how to import the required connection from the already existing configuration file.

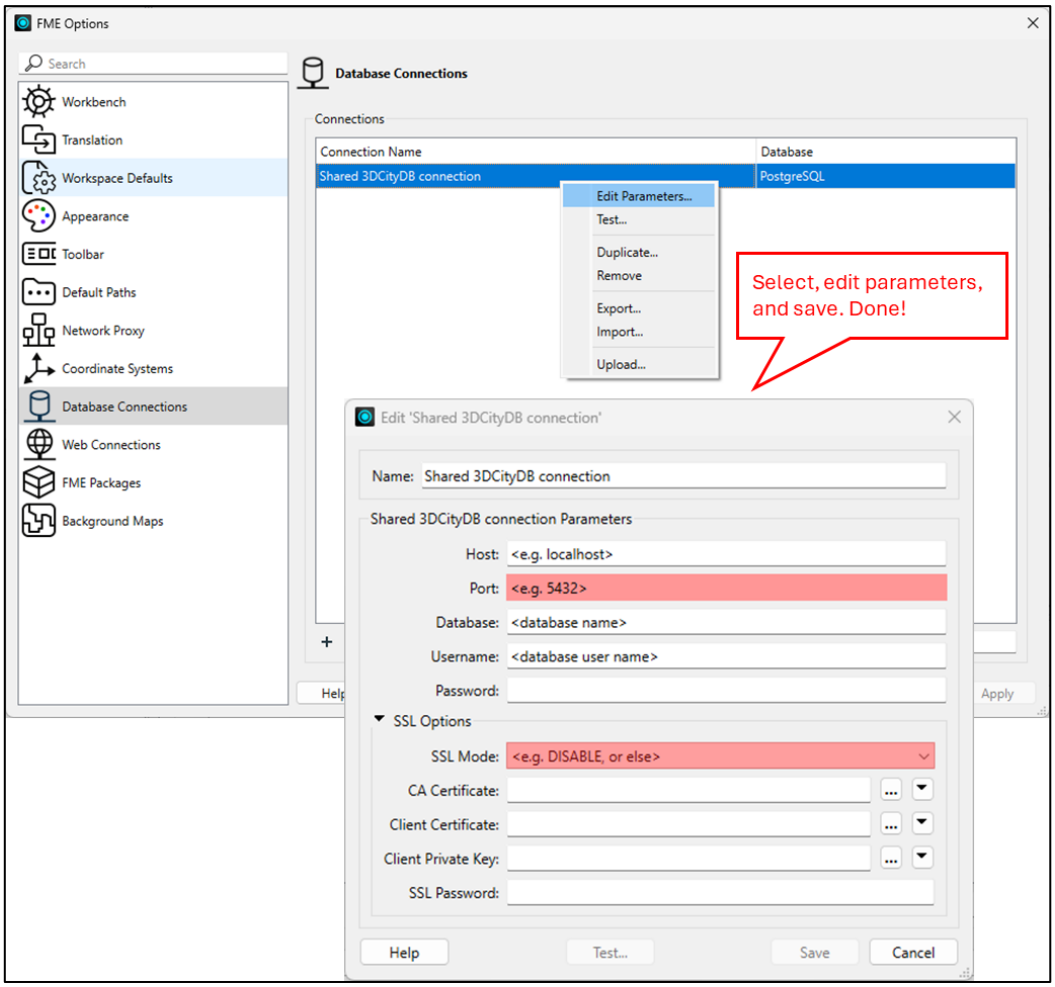


Figure 103. Example of how to set and store the connection parameters to the 3DCityDB server and database.

8.3 Running the workbench

It is now possible to run the FME workbench. Due to the size of the workbench, it is important that the "**Enable feature caching**" option is **not checked** (i.e. **disabled)**. This option can be set in the menu Run (or by opening the Run icon sub-options, as shown in Figure 104). Otherwise, depending on the size of the input data, the workbench might stop suddenly with a generic error. This most likely means that the available memory for caching data has been exhausted.

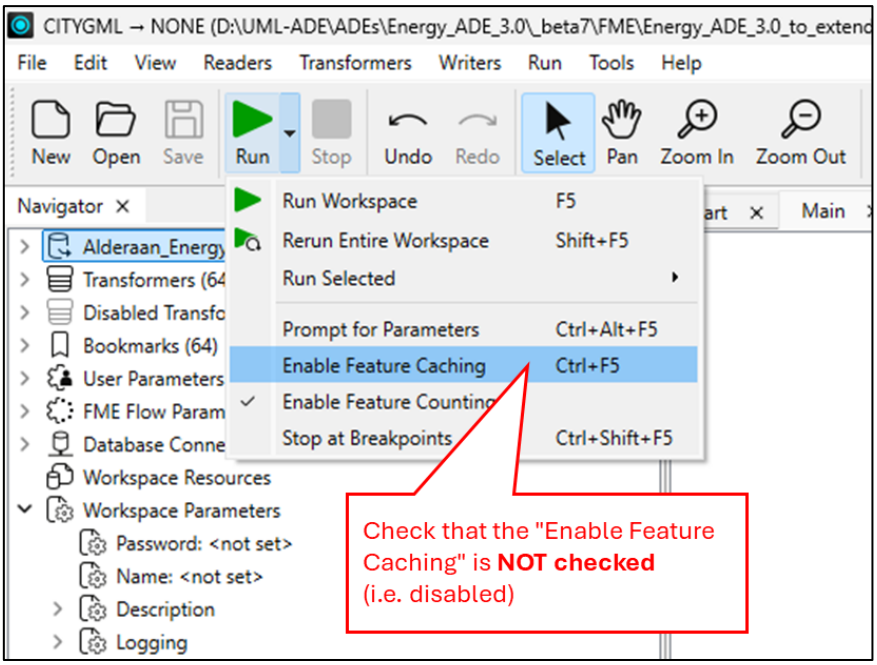


Figure 104. Example of how to check whether the Feature Caching is enabled or not.

Upon successful termination of the Energy ADE 3.0 data import, the user will be notified with a message in the FME Data Inspector (see Figure 105). A screenshot of an excerpt of the workbench is provided in Figure 106. As can be seen, there is a **READ ME** section providing a summary of the requirements to run the workbench – the same that are described in this section.

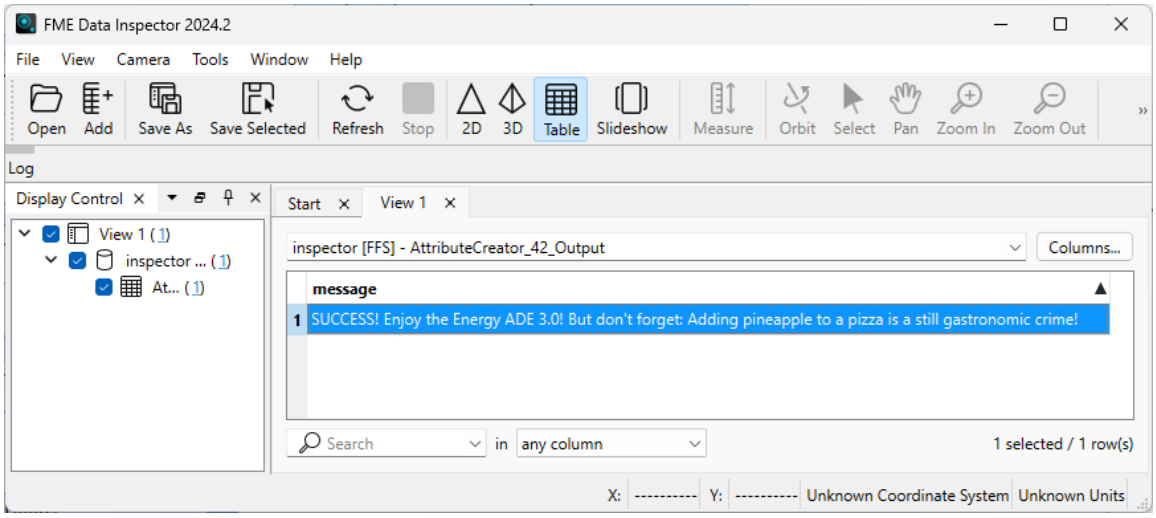


Figure 105. Example of message visualised upon completion of the workbench run.

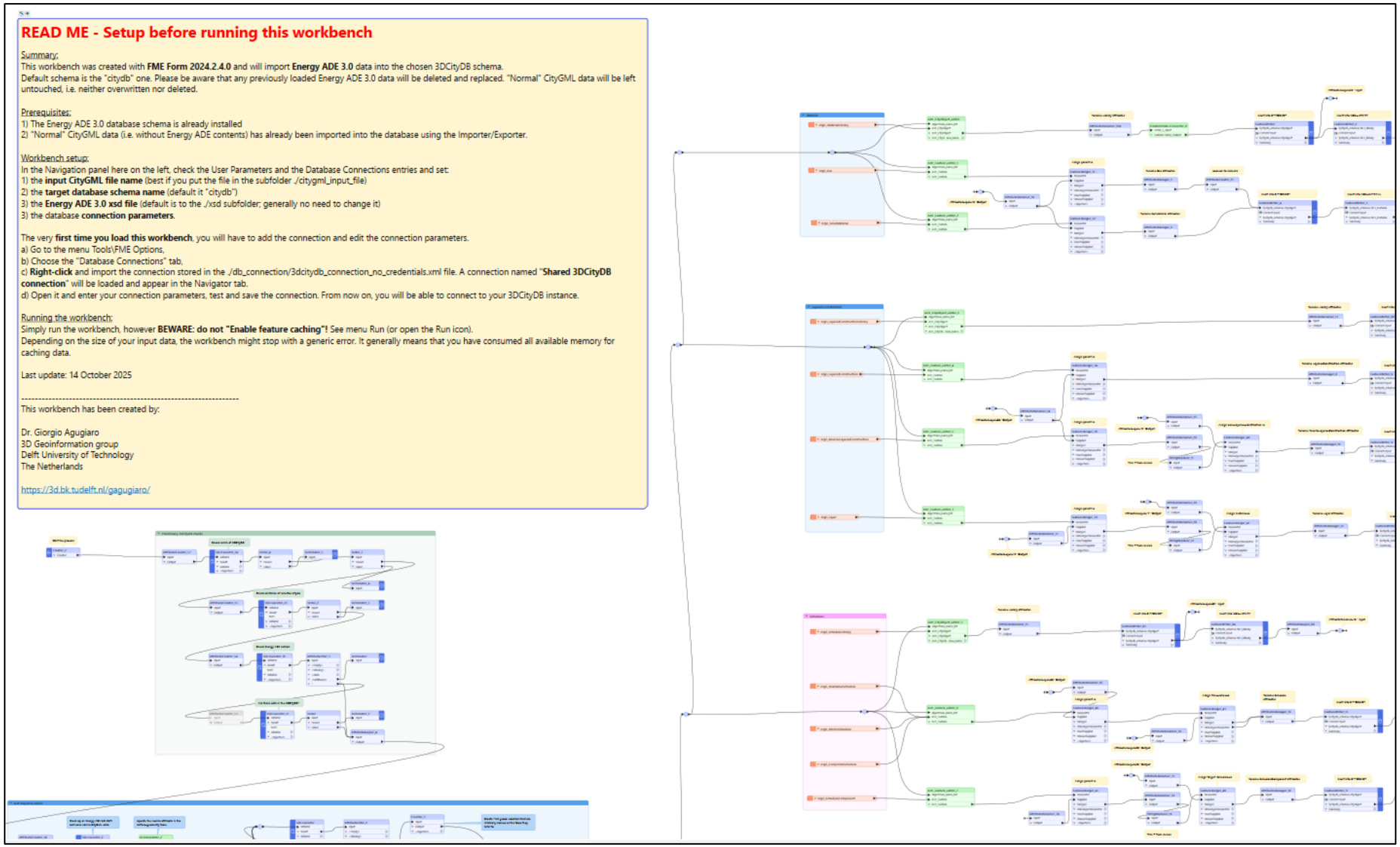


Figure 106. Screenshot of an excerpt of the FME Form workbench that allows to import Energy ADE 3.0 data into the 3D City Database.

9 Java libraries

As previously mentioned in section 2.2, adding full support to 3DCityDB and its Importer/Exporter tool for the Energy ADE 3.0 involves a two-step process. First, the ADE must be registered in the 3DCityDB instance (first-level extension). Once registered, the Importer/Exporter tool must be extended to support the ADE data import/export (second-level extension).
The Importer/Exporter tool provides a Java API that allows developers to implement ADE-specific import/export functionality for the second-level extension, through a simple plugin mechanism. This process includes two key steps:

- Creating an ADE module for **citygml4j**: This module is used by the Importer/Exporter to parse and write CityGML data, including ADE-specific elements.
- Implementing the **ADE API for the Importer/Exporter**: This involves writing code for reading from and writing to the ADE-specific tables in the database.

9.1 citygml4j module for Energy ADE 3.0

The citygml4j Java library is an open-source API designed to facilitate working with CityGML. It simplifies the process of reading, processing, and writing CityGML datasets, allowing developers to easily integrate CityGML data into software applications. citygml4j binds the XML schema definitions of CityGML to a structured Java object model. CityGML instance documents are deserialized into a hierarchy of corresponding Java objects that represent the content and structure of the document. This XML binding approach allows Java developers to work with CityGML data without needing to understand XML or low-level XML processing APIs. After processing, the Java content tree can be easily serialized back into a CityGML instance document. The Energy ADE 3.0 module is a custom extension of citygml4j that allows the library to handle the Energy ADE 3.0 schema, providing the following capabilities:

- **Parsing ADE elements:** It enables citygml4j to recognize and parse additional energy-related elements such as building energy usage, renewable energy systems, and energy systems from CityGML files.
- **Writing ADE elements:** The module also allows citygml4j to serialize these ADE elements back into valid CityGML files that conform to the Energy ADE 3.0 schema.
- **Schema Mapping**: It includes mappings between the Energy ADE 3.0 classes and the XML schema, ensuring correct interpretation and handling of these elements.

Without this module, citygml4j cannot process energy-specific attributes, making it impossible to fully manage Energy ADE 3.0 datasets in workflows, such as using the 3DCityDB Importer/Exporter for energy data.
The citygml4j module for Energy ADE 3.0 is currently under development within the DigiTwins4PEDs consortium, the main developer being HFT Stuttgart. All resources are available on their specific GitLab page[32].

[32] https://transfer.hft-stuttgart.de/gitlab/energy-ade/energy-ade-3-releases

9.2 Energy ADE 3.0 API for Importer/Exporter

The ADE API for the Importer/Exporter tool extends its functionality to support seamless reading and writing of Energy ADE 3.0 data within the 3DCityDB. This API integrates with the Importer/Exporter, allowing users to interact with ADE-specific data elements, ensuring that Energy ADE 3.0 datasets are processed and stored in 3DCityDB following CityGML standards. De facto, when fully implemented, it will replace the role of the FME workbench described in section 8. A prerequisite for implementing this API is the previously mentioned citygml4j module, which provides the necessary classes and methods for handling Energy ADE 3.0 data structures.
The Energy ADE 3.0 API for Importer/Exporter is currently under development within the DigiTwins4PEDs consortium, the main developer being HFT Stuttgart.

9.2.1 Supported Energy ADE 3.0 modules

At the time of writing (autumn 2025), the Java libraries for the 3DCityDB Importer/Exporter support the following modules and classes:

- **Data types** module (partial). Implemented classes:
    - AbstraftQualifiedAttribute and its subclasses
- **Time series** module (all classes)
- **Resource** module (all classes)
- **Urban function areas** module (all classes)
- **Weather station** module (all classes)
- **Core** module (partial). Implemented classes:
    - ADE *_CityObject*
    - ADE *_AbstractBuilding*
    - *RefurbishmentMeasure*
    - *WheatherData*

This means that if CityGML datasets contain only data relating to these Energy ADE 3.0 modules and classes, they can already be imported and exported from/to the 3DCityDB using the Importer/Exporter, and there is no need to use the FME Workbench described in the previous section 8.

10 QGIS plugin

The 3DCityDB Tools[33] plugin for QGIS[34] (Agugiaro et al., 2024) facilitates the management and visualisation of data stored in the CityGML 3D City Database 4.x, which supports CityGML v. 1.0 and 2.0. The plug-in allows to connect to local or remote instances of the free and open-source CityGML 3D City Database for PostgreSQL/PostGIS and to load data as "classical" layers into QGIS. Once data layers are available in QGIS, the user can interact with them as usual, i.e. perform analyses, work with associated attributes, explore and visualise the data in 2D and 3D, etc. Additionally, data in the database can be deleted, either using classical QGIS editing tools, or bulk-wise. A visual example is provided in Figure 107.
As semantic 3D city models tend to be huge datasets and are generally best managed in spatial databases, the main idea behind the development of this plug-in is to facilitate access and use of CityGML/CityJSON data for those practitioners that lack a deep knowledge of the international standard OCG CityGML data model, and/or have limited experience with SQL/Spatial-RDBMSs in general.

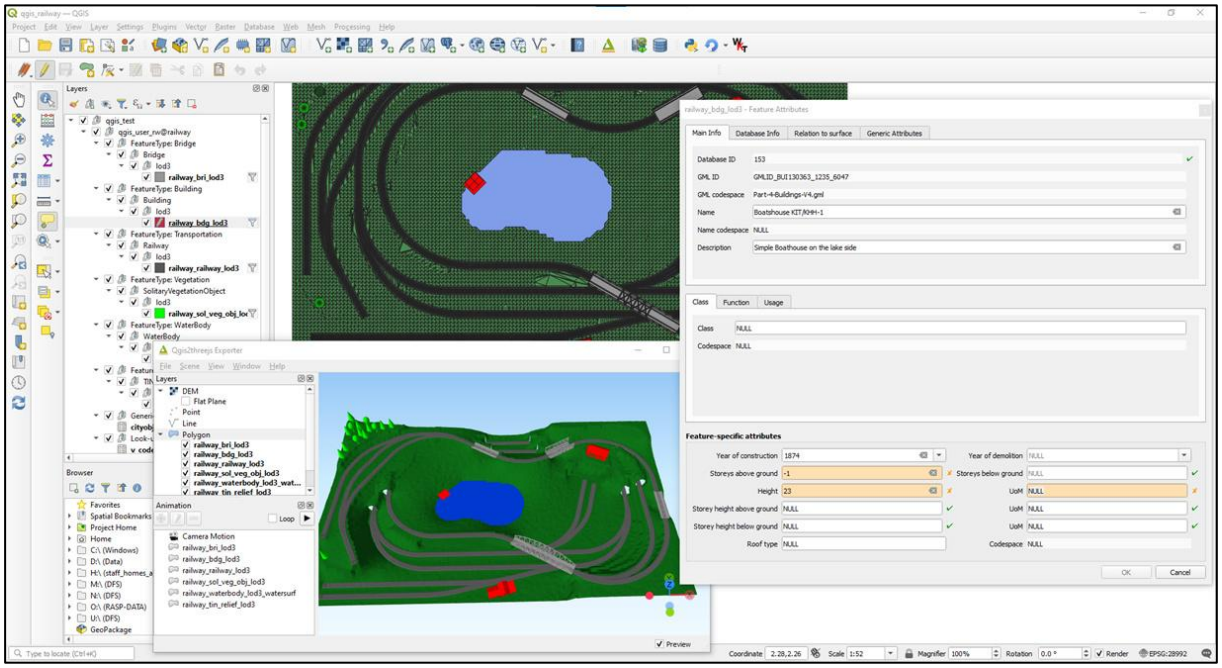


Figure 107. Example of the 3DCityDB-Tools plugin in QGIS. CityGML data is visualised as layers, and attributes can be retrieved and edited by means of specific attribute forms. Image source: https://github.com/tudelft3d/3DCityDB-Tools-for-QGIS

With the Energy ADE 3.0 reaching a sufficient level of stability and maturity, it is planned to extend the 3DCityDB-Tools plugin to support ADEs, and more specifically, to support the Energy ADE 3.0. This will allow the users to access, read (and edit, if allowed) CityGML data

[33] https://github.com/tudelft3d/3DCityDB-Tools-for-QGIS
[34] https://plugins.qgis.org/plugins/citydb-tools

with Energy ADE 3.0 contents directly from the database. The development will be based on preliminary work carried out during a MSc thesis in Geomatics ad TU Delft (Mbwanda, 2023).

11 Mapping Energy ADE 1.0 (KIT profile) data to Energy ADE 3.0

One of the goals of the Energy ADE 3.0 is to keep compatibility with data created with the Energy ADE 1.0, however the compatibility is limited to the KIT profile. The reason is that only the Energy ADE 1.0 KIT profile has been further developed (in terms of other software tools) and extensively used in the past years.
This section focuses therefore on providing some additional, specific notes to convert CityGML 2.0 + Energy ADE 1.0 (KIT profile) data to CityGML 2.0 + Energy ADE 3.0 data. It is recommended that the reader refer to the specific sections about the Energy ADE 3.0 conceptual data model in section 3 (in particular, each section about "Main changes from the Energy ADE 1.0") as relevant information is provided about the main differences between the two Energy ADE versions.

11.1 General mapping rules

In general, any data following the Energy ADE 1.0 KIT profile can be converted to Energy ADE 3.0 without any loss of data. However, some data transformations may be required. This refers to the following overarching operations:

- Renaming of class properties or of classes
- Adjusting values in codelists
- Converting enumerations to codelists
- Mapping of properties from one class to another
- Mapping of classes to classes of another module
- A combination thereof.

For the sake of simplicity, the following, specific conversion notes are grouped according to the **Energy ADE 1.0 KIT profile modules** and can be seen as a compendium of what already described in section 3.

11.2 Core module

All classes and all properties of the Core module can be mapped lossless. Some specific points of attention are listed in Table 5.

Table 5. Mapping notes for the Core module of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Conversion to Energy ADE 3.0
ADE *_AbstractBuilding*	Property *referencePoint* moved to class ADE *_CityObject*
EnergyDemand	The class is replaced by class *Energy* (in Resources module). Property *demandedBy* is replaced by property *operationType* of class *AbstractResource*

11.3 Building physics module

All classes and all properties of the Building physics module can be mapped lossless. Some specific points of attention are listed in Table 6.

Table 6. Mapping notes for the Building physics module of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
ThermalZone	Property *volumeGeometry* must be mapped to one of *lod{1,2,3}Solid* properties of class *AbstractBuildingPartition*. Properties *floorArea* and *volume* must be mapped to the analogous properties of class *AbstractBuildingPartition*.
ThermalBoundary	Based on the *thermalBoundaryType* property, objects must be mapped to the corresponding building thematic surface. Please refer to section 4.3 for more details. Relation *delimits* to class *ThermalZone* is replaced by association class *CityObjectRelation*. It can be used, in this case, to make the adjacency relation between two thermal boundaries explicit. Property *construction* is moved to property l*ayeredConstruction* of class ADE *_CityObject*. Property *surfaceGeometry* must be mapped to one of *lod{1,2,3}MultiSurface* properties of class *_BoundarySurface*.
ThermalOpening	Property *construction* is moved to property *layeredConstruction* of class ADE *_CityObject*. Property *surfaceGeometry* must be mapped to one of *lod{1,2,3}MultiSurface* properties of class *_BoundarySurface*.

11.4 Occupants' behaviour module

All classes and all properties of the Occupants' behaviour module can be mapped lossless. Some specific points of attention are listed in Table 7.

Table 7. Mapping notes for the Occupants' behaviour module of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
UsageZone	Property *floorArea* must be mapped to the analogous property of class *AbstractBuildingPartition*.
Occupants	Complex property *heatDissipation* has been "flattened" to the four *heatDissipation** properties.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
Facilities	This class has been reworked and moved to Devices module. The corresponding new class is *GenericDevice*. Complex property *heatDissipation* has been "flattened" to the four *heatDissipation** properties of class *AbstractDevice*.
DHWFacilities	This class has been reworked and moved to Devices module. The corresponding new class is *Boiler*.
LightingFacilities	This class has been reworked and moved to Devices module. The corresponding new class is *LightingDevice*.
ElectricalAppliances	This class has been reworked and moved to Devices module. The corresponding new class is *GenericElectricalDevice*.

11.5 Construction and material module

Despite the changes in the overall modelling rules of data belonging to this module (please refer to section 4.8.3) all classes and all properties of the Construction and material module can be mapped lossless. Some specific points of attention are listed in Table 8.

Table 8. Mapping notes for the Construction and material module of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
Construction	This class has been renamed to *LayeredConstruction*. Complex property *opticalProperties* has been "flattened" to the three properties *emissivity*, *reflectance* and *transmittance*. Property *glazingRatio* has been moved from class *OpticalProperties* to class *LayeredConstruction*.
Layer	Due to dropping class *LayerComponent*, property *thickness* is moved to class *Layer*.
LayerComponent	This class has been dropped. In case of *LayeredComponent* objects where the *areaFraction* is different from the default value of 1.0, then each *LayerComponent* must be assigned to its own *Layer* and its own *LayeredConstruction*. It is now possible to model set of layers that change *only* in terms of thickness over the whole surface associated to the same *LayeredConstruction*. If this is not the case, then different *LayeredConstruction* objects must be assigned each one to a different, separated portions of a surface.

11.6 Supporting classes module: Schedules

Despite the changes in the overall modelling rules of data belonging to this module (please refer to section 4.10) all schedule classes can be mapped lossless. Some specific points of attention are listed in Table 9.

Table 9. Mapping notes for the schedule classes of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
DailyPatternSchedule	This class has been dropped and replaced by class *CompositeSchedule*.
PeriodOfYear	This class has been dropped and replaced by class *ScheduleComponent*.
DailySchedule	This class has been dropped and replaced by class *AtomicSchedule* (having time series data), as child of a *ScheduleComponent* object.

11.7 Supporting classes module: TimeSeries

All time series classes can be mapped lossless. Some specific points of attention are listed in Table 10.

Table 10. Mapping notes for the time series classes of Energy ADE 1.0 KIT profile.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
AbstractTimeSeries	The datatype class *variableProperties* has been dropped and its properties have been assigned directly to class *AbstractTimeSeries*. In particular, properties *acquisitionMethod*, *interpolationType* and *source* are now properties of *AbstractTimeSeries*. Properties *qualityDescription* and *thematicDescription* are merged and replaced by property *description* inherited from *_Feature*.

11.8 Supporting classes module: WeatherData

All weather-related classes can be mapped lossless. Some specific points of attention are listed in Table 11.

Table 11. Mapping notes for the weather classes of Energy ADE 1.0 KIT profile

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
WeatherStation	Properties *stationName* and *position* have been dropped, as they are replaced by properties *name* and *referencePoint* inherited from classes *_Feature* and ADE *_CityObject*, respectively.

Energy ADE 1.0 KIT profile class	Mapping to Energy ADE 3.0
	Property *parameter* has been dropped, as redundant due to property *weatherData* inherited from class ADE *_CityObject*. Please note: this was an error in the Energy ADE 1.0 that has been finally fixed.

References

Agugiaro, G., Benner, J., Cipriano, P., Nouvel, R., 2018. **The Energy Application Domain Extension for CityGML: Enhancing interoperability for urban energy simulations**. *Open Geospatial Data, Software and Standards*, 2018 3:2. SpringerOpen, United Kingdom. https://doi.org/10.1186/s40965-018-0042-y

Agugiaro, G., Pantelios, K., León-Sánchez, C., Yao, Z., Nagel, C., 2024. **Introducing the 3DCityDB-Tools plug-in for QGIS**. *Recent Advances in 3D Geoinformation Science - Proceedings of the 18th 3D GeoInfo Conference*, Springer, pp. 797–821. https://doi.org/10.1007/978-3-031-43699-4_48

Akahoshi, K., Ishimaru, N., Kurokawa, C., Tanaka, Y., Oishi, T., Kutzner, T., and Kolbe, T. H., 2020. **i-Urban Revitalization: conceptual modeling, implementation, and visualization towards sustainable urban planning using CityGML**. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., V-4-2020, 179–186, https://doi.org/10.5194/isprs-annals-V-4-2020-179-2020

Bachert, C., León-Sánchez, C., Kutzner, T., Agugiaro, G., 2024. **Mapping the CityGML Energy ADE to CityGML 3.0 using a model-driven approach**. *ISPRS Int. Journal of Geo-Information*, MDPI, 2024, 13(4). https://doi.org/10.3390/ijgi13040121

Chaturvedi, K., Kolbe, T.H., 2016. **Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities**. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-2/W1, ISPRS. https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016

Van den Brink, L., Stoter, J., Zlatanova, S., 2013. **UML-Based Approach to Developing a CityGML Application Domain Extension: UML-Based Approach to Developing a CityGML Application Domain Extension**. *Trans. GIS* 2013, 17, 920–942. https://doi.org/10.1111/tgis.12026

Gröger, G., Plümer, L., 2012. **CityGML—Interoperable semantic 3D city models**. *ISPRS J. Photogramm. Remote Sens*. 2012, 71, 12–33. https://doi.org/10.1016/j.isprsjprs.2012.04.004

Kutzner, T., Chaturvedi, K., Kolbe, T.H., 2020. **CityGML 3.0: New Functions Open Up New Applications**. *PFG—J. Photogramm. Remote Sens. Geoinf. Sci*. 2020, 88, 43–61. https://doi.org/10.1007/s41064-020-00095-z

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

Mbwanda, T., 2023. **Further Development of a QGIS plugin for the CityGML 3D City Database**. MSc thesis in Geomatics, Delft University of Technology, The Netherlands.

Padsala, R., Gebetsroither-Geringer, E., Peters-Anders, J., Coors, V., 2021. **Inception of harmonising data silos and urban simulation tools using 3d city models for sustainable management of the urban food water and energy resources**. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., VIII-4/W1-2021, 81–88, https://doi.org/10.5194/isprs-annals-VIII-4-W1-2021-81-2021

Open GeoSpatial Consortium, 2021. **OGC City Geography Markup Language (CityGML) 3.0 Conceptual Model Users Guide.** https://docs.ogc.org/guides/20-066.html

Yao, Z., 2020. **Domain Extendable 3D City Models – Management, Visualization, and Interaction**. PhD thesis. Technische Universität München. https://mediatum.ub.tum.de/doc/1454692/1454692.pdf

Document version

Version	Date	Author	Notes
1.0 draft	2 June 2025	G. Agugiaro	Added section on Java libraries Added section on QGIS plugin
1.0 draft	6 June 2025	G. Agugiaro	Added section on Mapping Energy ADE 1.0 data to Energy ADE 2.0.
1.0	7 June 2025	G. Agugiaro	First release as version 1.0
1.0.1	10 June 2025	G. Agugiaro	Fixed typos, formatting and minor layout errors
1.0.2	13 June 2025	G. Agugiaro	Fixed typos, formatting and minor layout errors. Extended/improved Database section Extended/improved FME workbench section
1.0.3	20 June 2025	G. Agugiaro	Fixed typos and other minor layout issues.
1.0.4	25 June 2025	G. Agugiaro	Adapted text and images to latest bugfix of Energy ADE 2.0 beta 7. Added further details to modelling rules of *UrbanFunctionArea* objects.
1.0.5	30 June 2025	G. Agugiaro	Added copyright statement (CC BY-NC-SA) and prepared for release in Energy ADE GitHub repo.
1.1.0	26 October 2025	G. Agugiaro	Adapted text, images, etc. for version change from 2.0 to 3.0.

This is the last page of the document. This page is left blank on purpose.