DOI: 10.1111/tgis.12723

RESEARCH ARTICLE

Transactions in GIS WILEY

A modular graph transformation rule set for IFC-to-CityGML conversion

Helga Tauscher[1] | Joie Lim[2] | Rudi Stouffs[2]

[1]Faculty of Spatial Information, HTW Dresden - University of Applied Sciences, Dresden, Germany

[2]Department of Architecture, National University of Singapore, Singapore, Singapore

Correspondence
Helga Tauscher, Faculty of Geoinformation, Dresden University of Applied Sciences, Friedrich-List-Platz 1, Dresden, Germany.
Email: helga.tauscher@htw-dresden.de

Funding information
Virtual Singapore, Grant/Award Number: NRF2015VSG-AA3DCM001-008

Abstract

Conversion of Industry Foundation Classes (IFC) building models into CityGML city models is one of the operational scenarios for BIM–GIS integration, with a variety of applications producing and consuming data on either side. Given the in-depth cross-domain knowledge required to specify such conversions, the heterogeneity of the IFC input data and the use cases for the resulting CityGML, flexible and configurable solutions are needed that make conversion details accessible to domain specialists. Graph transformation as a conversion method fulfils these requirements. We propose to extend the modularity given by single transformation rules at a more coarse-grained level and identify four layers with modules of associated rules. We describe a self-contained set of rules across these modules and demonstrate its application to a range of building models.

1 | INTRODUCTION

1.1 | Background and objectives

The findings reported here derive from a research project with the aim of developing a methodology and algorithms for the automated conversion of building models in the Industry Foundation Classes (IFC) data format into 3D city models in the Geographic Markup Language (GML) application schema CityGML, capturing both geometric and semantic information as available in the building model, including exterior as well as interior structures such as corridors, rooms, internal doors, and stairs (Stouffs, Tauscher, & Biljecki, 2018). While automation from IFC to CityGML has been researched and demonstrated before (e.g., de Laat & van Berlo, 2011; Deng, Cheng, & Anumba, 2016; Donkers,

Ledoux, Zhao, & Stoter, 2016), mainly for exterior building shells, this research focused on achieving a mapping for both exterior and interior structures at level of detail (LOD) 3/4 with their spatial and non-spatial semantics, informed by potential use cases. Our research particularly targeted an approach and methodology that can cater for the variety of IFC files produced by CAD software and the variety of use cases consuming the converted CityGML files.

Besides the automated conversion from IFC to CityGML, in particular from IFC4 to CityGML 3.0, the research project considered two accompanying questions. First, how is the IFC exported from native building information model (BIM) software as input to the conversion? And second, how is the converted CityGML used afterwards? These parts of the research project led to considerations and guidelines for authoring and exporting the building model and to an application domain extension for CityGML, geared to allow for the representation of information beyond what CityGML currently provides for. In this article we focus on the conversion from IFC to CityGML and the methodology we adopted for this purpose, which provides the flexibility to accommodate results from the accompanying research.

The variety on both the input side (building model authoring and export to IFC) and on the output side (use cases relying on converted CityGML) requires a flexible conversion process that is able to adapt and can be configured. In addition, to describe the variety and determine its impact on the conversion, the conversion execution needs to be informed by the knowledge of domain experts: building model authors and users of the generated city models alike.

The research also sought a way to indicate a measure of completeness of a conversion, both in terms of its configuration and its application to a particular input data set.

1.2 | Methodology

We chose graph transformation as a paradigm because graphs and graph transformations provide a well-understood rigorous formal model for the conversion, while the transformation configuration is specified in a declarative way and has a sensibly intuitive graphical representation.

Graphs are recognized as suitable structures to represent many types of subject-matter in computer science, for example abstract syntax trees, dependency graphs and object graphs. Figure 1 shows the object graph of a partial IFC file (Listing 1) on the left and the object graph of a partial CityGML file (Listing 2) on the right. Each object (entity in IFC) is represented as a node in the graph and as a box in the diagram. The first line of the box label

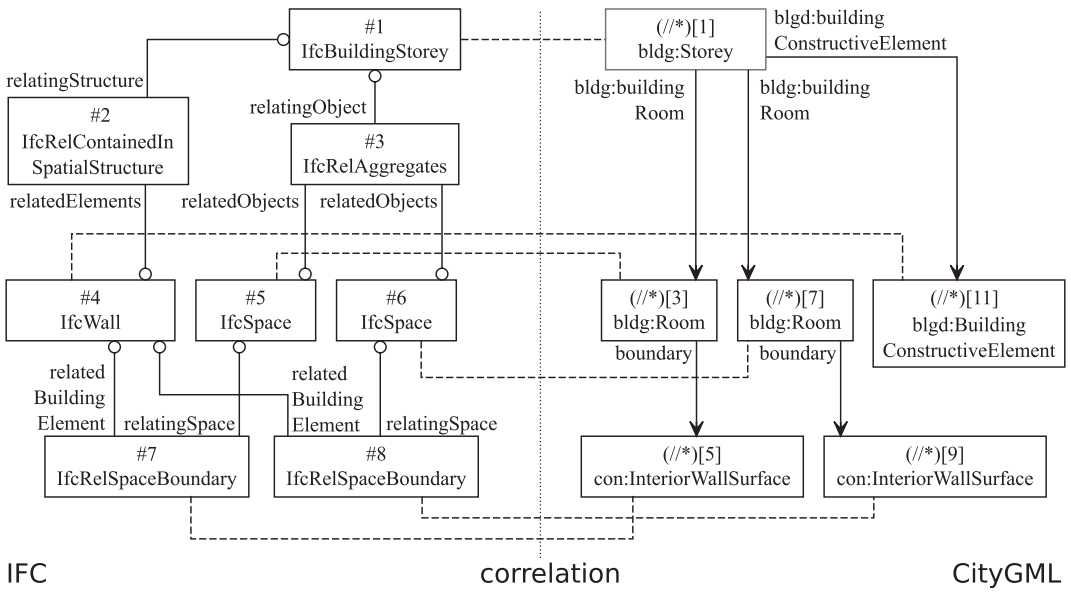


FIGURE 1 Triple graph for the spatio-semantic structure of a building, consisting of an IFC graph (left), a CityGML graph (right), and a correlation graph (dashed lines)

denotes a local identifier of the entity or object represented. In the diagram, IFC entities are identified by the entity number, CityGML objects by an XPath expression for the position in the XML tree (depth-first enumeration).

Likewise the IFC and the CityGML schemas can be represented as type graphs. The relation between instance and schema is captured in a typed object graph, where a graph morphism maps the nodes and edges of an object graph to nodes and edges of a type graph. For the diagrams in this article, we follow the convention to integrate the depiction of object and type graph together with the morphism by labelling the nodes and edges with their

```
#1=IfcBuildingStorey(...);
#2=IfcRelContainedInSpatialStructure(...,(#4),#1);
#3=IfcRelAggregates(...,#1,(#5,#6));
#4=IfcWall(...);
#5=IfcSpace(...);
#6=IfcSpace(...);
#7=IfcRelSpaceBoundary(...,#5,#4,...);
#8=IfcRelSpaceBoundary(...,#6,#4,...);
```

LISTING 1. IFC sample

```
<bldg:Storey>                                    <!--  1  -->
    <bldg:buildingRoom>
        <bldg:BuildingRoom gml:id="r1">          <!--  3  -->
            <boundary>
                <con:InteriorWallSurface />      <!--  5  -->
            </boundary>
        </bldg:BuidlingRoom>
    </bldg:buildingRoom>
    <bldg:buildingRoom>
        <bldg:BuildingRoom gml:id="r2">          <!--  7  -->
            <boundary>
                <con:InteriorWallSurface />      <!--  9  -->
            </boundary>
        </bldg:BuidlingRoom>
    </bldg:buildingRoom>
    <bldg:buildingConstructiveElement>
        <bldg:BuildingConstructiveElement>       <!-- 11  -->
    </bldg:buildingConstructiveElement>
</bldg:Storey>
```

LISTING 2. CityGML sample

respective types.[1] A more detailed description of the mathematical model and diagrammatic representations is given by Tauscher and Crawford (2018) and Tauscher (2019).

Figure 1 contains a third graph, the correlation graph (dashed lines), whose edges connect pairs of nodes from the two typed graphs. Such a triple graph represents an instance of consistently correlated IFC and CityGML data. Triple graph grammars (TGGs), as introduced by Schürr (1995), describe languages of triple graphs through a set of production rules. This way they provide a formalism to specify correlations between graph-like data structures. Operational graph transformation systems derived from such a grammar allow these potentially disparate structures to be integrated and transformed. We adopted a TGG to formally relate IFC and CityGML, both semantically and geometrically, and developed a forward transformation system to transform a building information model, expressed as an IFC object graph, into part of a city model expressed as a CityGML object graph.

A graph transformation system consists of transformation rules, where each rule denotes a state of a particular part of the triple graph before and after application of this rule. The states themselves are given as two triple graphs, which we call template graphs. A graph transformation engine operates on a triple graph such as the one in Figure 1. To apply a particular rule, the graph transformation engine looks up a match for the rule's "before" template graph within the triple graph and, consequently, modifies the triple graph to contain a match for the "after" template graph instead. This match-and-replace operation on the triple graph is called a "rule application". We represent the rules' before and after templates in a common way, as an integrated diagram, where the state before application (also called context) is given in mid-grey and the difference with respect to the state after is given as additions in darker grey and deletions in lighter grey. Examples can be found throughout the article and in the Appendix A.

To perceive the conversion process as a graph transformation system allows domain specialists to focus on the desired result of the conversion instead of the steps to take to arrive at that result. The rules capture the variable parts of the conversion with regard to the domain models and semantics in a declarative way. Thus, rule engineers do not have to bother with the algorithms for carrying out the conversion. Visual representations are supposed to facilitate imagination and support the processing of abstract issues such as the conversion process, especially among engineers who are used to assisting their thinking and problem-solving processes with diagrams and sketches.

Graph theory and graph-based methods have been used elsewhere to handle certain aspects of IFC. For instance, Sun, Liu, Gao, and Han (2015) use an IFC graph as the basis for a compression algorithm, and Khalili and Chua (2015) try to use graph methods for topological IFC queries. Tauscher, Bargstädt, and Smarsly (2016) suggest employing Dijkstra's shortest path algorithm to simplify queries. Ismail, Nahar, and Scherer (2017) store IFC data in a neo4j database for further processing. Beyond IFC, graphs are also employed for BIM data in general; for instance, Vilgertshofer and Borrmann (2017) apply graph rewriting to generate parametric infrastructure models.

On the CityGML side, Yao and Kolbe (2018) use graph transformation to derive the schema for a relational database management system from the CityGML XML Schema Definition. Nguyen, Yao, and Kolbe (2017) use a graph database to compare city models based on the graph representation. To our knowledge, however, graph transformation has not been employed for model transformation between the two domains of building information and city modelling.

We describe the rule set we have developed in detail, the process of development and the application to a range of sample data sets. Beyond a proof of concept for the graph transformation approach, we thus provide an in-depth insight into the conversion as such. In Section 2 we elaborate on the development of the rule set and the considerations for a modular design. Sections 3–6 are dedicated to the four layers of the rule set. In Section 7 we show the results of applying the rule set to five different data sets. Finally, in Section 8, we discuss the results, limitations of the approach and future work.

In Sections 3–6 we systematically explore and cover the space of potential mappings and rules by working through independent areas of the conversion process as reflected in layers. These sections yield a reduced but self-contained exemplary rule set, which can be used as a blueprint or guidance even when implementing IFC-to-CityGML conversion with another approach. In general, we assume basic familiarity with both IFC and CityGML on the part of the reader. We refer to the seminal introductory literature for fundamentals, and to the specifications of the standards for details. In particular, for Section 5 on geometry we do not repeat the IFC basics but rather share conversion details that we deem potential pitfalls.

1.3 | Authoring and execution environment

In the following, we briefly describe the authoring and execution context in which rules were developed and tested.

For provision of the IFC files we set up an instance of the Opensource BIMserver[2] (Beetz, van Berlo, de Laat, & van den Helm, 2010). BIMserver is responsible for parsing, storage and preprocessing of IFC files. This set-up allows for efficient access of even large files during the actual conversion and for shared provisioning of the input data during collaborative development and testing of the rule set.

To edit, maintain and deliver the rules and rule sets, we use a custom web application written in Groovy with the Grails framework (Smith & Ledbrook, 2014). This rule server application is described more thoroughly in Tauscher (2019). The application provides a tailored domain-specific language (DSL) to write and edit the rules, allows rules to be grouped in rule sets with the option to use a particular rule in multiple sets, and automatically generates diagrams for the rules. Most diagrams in this article stem from this application.

The actual conversion engine is implemented as a Java client. The client fetches a specified revision of a BIMserver project as IFC input and a specified rule set from the rule server application and carries out the conversion based on the IFC and the rule set input. The resulting CityGML is serialized using Java representations of the XML Schema generated with JAXB (Java Architecture for XML Binding).

2 | RULE SET DEVELOPMENT AND ORGANIZATION

Here we describe the development of the rule set and the considerations for a modular design of the rule set.

2.1 | Rule set development process

During the research we developed the rules iteratively, and in parallel to developing the tools, specifying and refining the transformation target and use cases, and gathering and analysing the input data. We used the latest version of IFC, IFC4 (ISO, 2013), and the current draft version of CityGML 3.0 (Kutzner, Chaturvedi, & Kolbe, 2020; Kutzner & Kolbe, 2018), with tool support still catching up and specifications evolving. Even with good suitability of graph transformation for iterative development, the task of keeping an overview of the rules, their variants and dependencies, turned out to be a challenge.

Anjorin, Leblebici, Kluge, Schürr, and Stevens (2015) acknowledge that transformation rule development (here for bi-directional transformation in general, focusing on synchronization) is a non-trivial task. They propose a set of guidelines, some of which resemble general software development best practices, such as iterative test-driven development and the choice of appropriate abstractions.

Retrospectively, we followed many of these guidelines intuitively. For example, we took a top-down approach and we used dedicated reduced sample data sets to employ test-driven development. However, we did not continuously test against consistent source and target models throughout the development process, mainly due to the lack of a clear target specification.

They further suggest, for instance, retaining as much complexity as possible within the TGG rules. Instead, we did consider the option of falling back on procedural rule specification through converters. Since the declarative description is much more tedious to achieve, it is tempting to express logic in converters instead of the TGG rules. This prevents static analysis of rules and makes conversion logic invisible to those domain specialists who understand only the conversion rules expressed in the domain-specific language or diagrams and not the procedural conversion logic expressed in a general purpose language source code. Thus, we strove to maintain a balance with as much conversion logic covered by rules, pulling procedural logic out of the converters and turning it into declarative specifications at a later stage if possible.

2.2 | From graph template to transformation rule

In the development process of graph transformation rules, the identification of subgraphs with well-defined semantics on the source and target side and the subsequent correlation of these potentially diverging semantic structures is a crucial step.

In order to identify and isolate the semantic aspects of the transformation, we adapt a process that starts with instance graphs and then proceeds from there to templates and rules. We first outline exemplary source and target instance graphs and then break these into smaller pieces by identifying similarities and differences between the various samples.

An example of larger instance graphs on the source and target side can be seen in the triple graph example in Figure 1. For a worked example of smaller, already decomposed instance graphs, which serve as building blocks for the subsequent construction of rules, see Sections 4.1 and 6.1.

This is similar to the approach taken by Varró (2006) who suggest specifying rules by example of the matching instances. Kindler and Wagner (2007) take the approach one step further and suggest a semi-automatic derivation of rules from exemplary instance graphs. However, we keep the rule creation process (e.g., the selection and correlation of the source and target instance pieces) manual.

2.3 | Modularity, dependencies and layers

The goal of modular rule organization is to facilitate reuse and to allow control of the conversion process at different levels of detail. These different levels require different levels of domain expertise and in-depth knowledge.

The most coarse-grained level of modularity is concerned with multiple rules. A configuration option for the conversion process (e.g., the choice of a solid or surface model target) may require a particular set of rules to be developed and used in the conversion process. Such a set of related rules, bound to a configuration option, bundles multiple rules that are often used together. This allows the set of rules to be selected at once rather than individually.

The next, medium level of modularity is given by the rule-based approach as such, because each rule can be reused in different configurations. A domain expert pulling together a rule set for a particular use case need not create new rules, nor edit rules and thus need not necessarily master the rule specification language or recall every subtle detail of the domain models.

Figure 2 shows these two levels of modularity. On the left, there are coarse choices for the conversion configuration. In the centre, rules in the repository are depicted as rows in tables, the options as columns. On the right, a particular configuration for the conversion is shown as a list of rules. The check marks and arrows represent the selection of the different rules to be included from the repository into the rule set. As we will later see, modularity is also desirable at a finer-grained level, below a single rule.

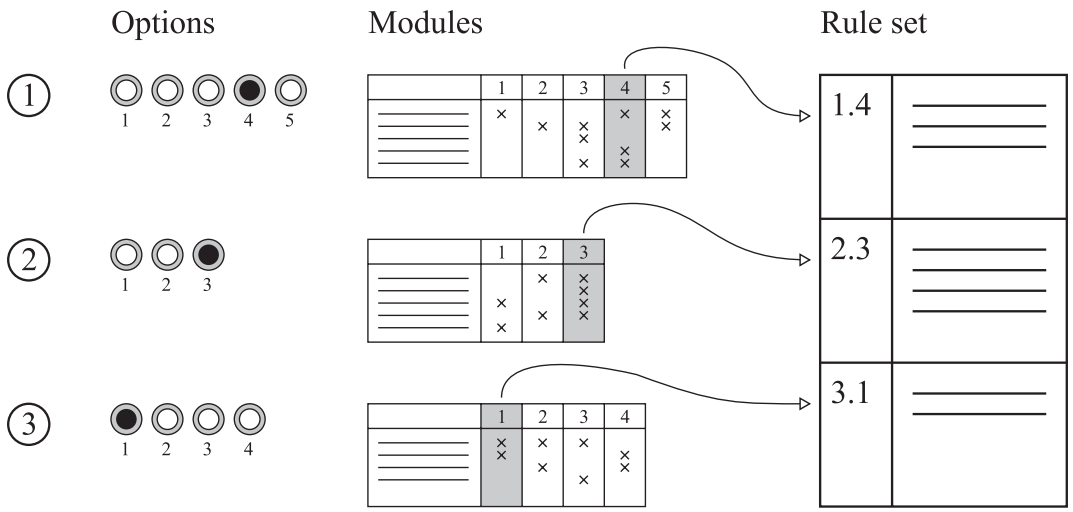


FIGURE 2 Configuration options, corresponding modules and selected rule sets, organized as groups

Semantically related rules (i.e., those belonging to the same option) often also depend on each other. A rule is said to depend on another rule if its successful application relies on the other rule being applied upfront. Ideally, we want to concentrate dependencies inside option modules and minimize dependencies between rules bound to different options, such that user choices are as independent as possible and dependencies between options are kept to a controllable level.

There is also the inverse case where application of a rule prevents successful matching of another rule (e.g., when the first rule deletes nodes required to match the context of the second rule). This has led to the concept of layered graph grammars, where rules are organized in layers according to the priority of their application. Otherwise rule application order is figured out by the graph transformation engine.

We are utilizing a layered approach to keep control over and minimize dependencies between options. Rules in one layer only depend on rules in higher-priority layers, not on those in lower-priority layers. An option-bound module only spans rules in one layer. During a conversion, a selection of rules from each layer is applied successively. We have identified four layers and organized our rules according to this structure. Selected modules from these layers and the rules they contain are described in more detail in the following sections: general structure layer (Section 3), spatio-semantic layer (Section 4), geometry layer (Section 5), and properties layer (Section 6). The rules shown in those sections form a complete and self-contained rule set which is suitable for application to common IFC input data sets. To facilitate potential implementation, the complete rule set is listed in the Appendix A.

3 | GENERAL STRUCTURE LAYER

The first layer contains rules that define the entry point and primary paths to follow on the source graph as well as the main structure to generate for the target graph. There are multiple ways to approach this for the IFC and CityGML case. We can enter the IFC graph through either the project's geometric context or the project's spatial hierarchy. The latter is more common and we show this option here. The first layer is responsible for traversing the main semantic elements, that is, IFC products and CityGML features.

The rules shown in Figure 3 create the spatial structure of one or more buildings in a project, further subdivided into parts and/or storeys. The rules of the following layers can subsequently rely on the spatial structure being already in place.

Since both IFC and CityGML provide for a hierarchical spatial decomposition of buildings, the rule construction consists merely of the correlation of similar decomposition levels. One difficulty is the varying granularity and flexibility of the decomposition structure. While IFC has a fine-grained and flexible structure, CityGML has a more coarse-grained and restricted structure. For instance, an IFC project can consist of multiple sites with multiple buildings on each site. Alternatively, a project can consist of one or more buildings that are not organized on a site, but directly assigned to the project. In contrast, a CityGML root model consists of one or more buildings without the intermediate site level. In IFC, each building and storey can again consist of buildings or storeys; the particular levels are then tagged as *complex*, *element* or *partial* buildings or storeys. This is not foreseen in CityGML. A transformation rule must thus collapse the fine-grained structure to a coarse-grained by skipping unmapped levels on the IFC side. Further, one rule is needed for every spatial decomposition pattern that we want to cover on the IFC side. Clearly, to match every possible combination requires many rules or is simply not possible.

4 | SPATIO-SEMANTIC LAYER

The spatio-semantic layer[3] comprises rules that bridge the semantic (structure and properties) and geometry layers. The geometry layer is concerned with those parts of the triple graph that carry any kind of spatial semantics. Geometries themselves appear as graph structures reflecting the composition of, for example, surfaces from curves

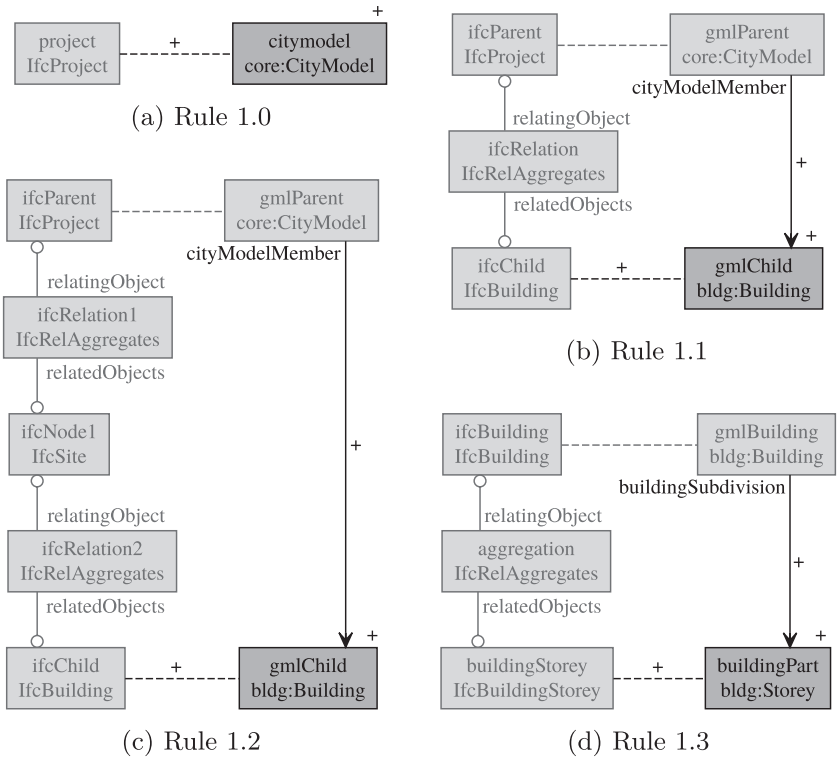


FIGURE 3 Rules for the spatial structure

and curves from points. Some of the geometry nodes are of an exclusively spatial nature, while others carry non-spatial meaning beyond the geometry. For example, a surface may be just a defining resource of a solid geometry or be treated as a relevant element in itself and charged with meaning. This is the purpose of the spatio-semantic layer, to identify these semantically relevant spatial elements and specify how they are embedded into the general structure.

4.1 | Dimensions of the spatio-semantic mapping

As described in Section 2.2, we develop rules by breaking instance graphs into smaller template graphs on the source and target side and combining the two to construct rules. Figure 5 shows such template graphs for the IFC side, Figure 6 for the CityGML side.

In doing so, we derived three essential characteristics of the spatio-semantic mapping, each one of which might appear in one of two options. The first characteristic relates to the source geometry and distinguishes preprocessed triangulated geometry from original IFC geometry. The second one relates to the main semantics of source and target geometry, whether it represents constructive building elements or their boundary representation. The third one relates to the target level of detail and projection, either LOD3/4 or LOD0 (floorplans). Thus, the space of mappings consists of three dimensions with two options each, resulting in a total of eight variants. Each variant can be described by a combination of three values d_0, d_1, d_2, one for each dimension, with value 0 or 1 depending on the characteristic of the respective dimension. In the following we number the variants according to these values as

$$\sum_{i=0}^{2} d_i 2^i + 1.$$

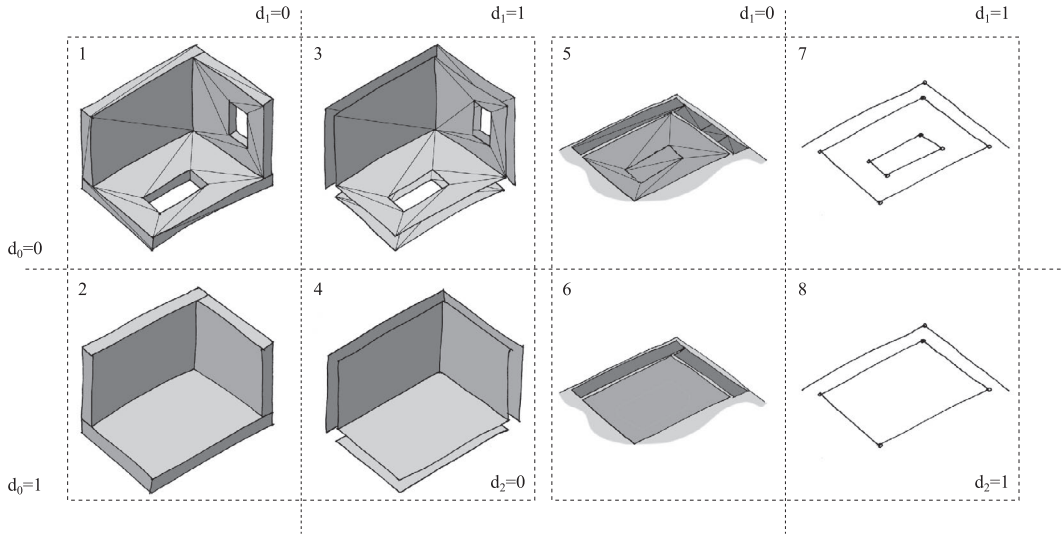


FIGURE 4 Spatio-semantic characteristics of rule sets: different input types in rows (triangulated or not), different output types in columns (solid elements, boundary surfaces, floorplans)

TABLE 1 Options choice for variant 6

	0	1
d_0	□ Preprocessed geometry	■ Original IFC geometry
d_1	■ Solid building elements	□ Boundary surfaces
d_2	□ LOD3 unprojected 3D	■ LOD0 floorplans

For example, if we use original IFC geometry ($d_0 = 1$), create constructive building elements ($d_1 = 0$) and project to a floorplan for LOD0 ($d_2 = 1$), we refer to it as variant 6 ($= 101_2 + 1$) (Figure 4). If we manage to transfer these aspects to well-defined subsets of the rules in the spatio-semantic layer, this would allow for a configuration at a higher level of granularity by selection of three options as discussed in Section 2.3. For the example above, this higher-level configuration of options and selections is shown in Table 1.

4.2 | Template graphs for the spatio-semantic mapping

In this section we identify respective template graphs for the three dimensions on the source (d_0, d_1) or target side (d_1, d_2).

The first dimension relates to the extraction of geometry from the IFC source and distinguishes:

- preprocessed triangulated geometry ($d_0 = 0$); and
- original IFC geometry representations ($d_0 = 1$).

Preprocessed triangulated geometry is the geometry that is generated during export from BIM authoring software or by an external process and stored in addition to the IFC with its original geometry. Such preprocessed geometry can be useful for efficient visualization implementation, for example. Figure 5b shows how the

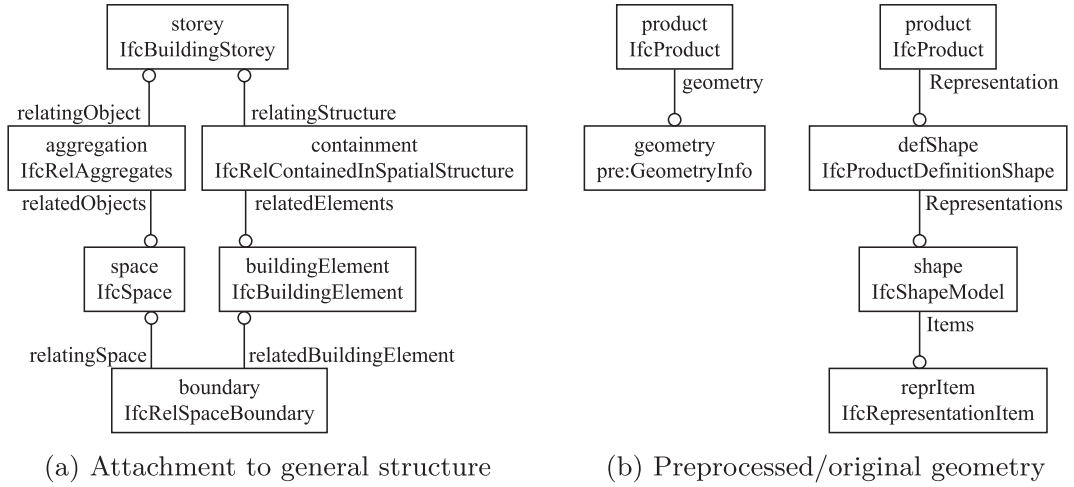


FIGURE 5 Spatio-semantic IFC source templates for building elements

preprocessed triangulated geometry (left, $d_0 = 0$) and the original IFC geometry (right, $d_0 = 1$) are embedded in IFC graphs for the case of $d_1 = 0$, that is, the spatio-semantic IFC source structure of products.

The second dimension relates to the main type of spatial semantics used, distinguishing:

- building elements ($d_1 = 0$); and
- boundary surfaces ($d_1 = 1$).

Both IFC and CityGML do in general support either option. Figure 5a shows a sample instance graph with the two types (`IfcSpace`/IfcBuildingElement versus IfcRelSpaceBoundary) for the IFC side. Figure 6a illustrates the case $d_1 = 0$, relating to constructive elements for the CityGML side. Similar instance graphs exist for the $d_1 = 1$ case on the CityGML side where the main spatial semantics appear as "semantic surfaces".

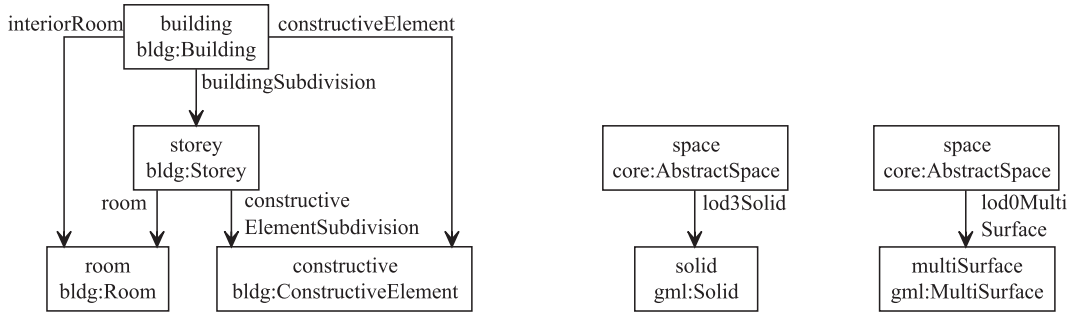
The third dimension relates to the target projection and level of detail, distinguishing:

- solids (3D) and surfaces (2D), $d_2 = 0$; and
- floorplan surfaces (2D) and curves (1D), $d_2 = 1$.

Figure 6b illustrates the spatio-semantic CityGML target structures, either for LOD3/4 with unprojected 3D/2D geometry (left, $d_2 = 0$) or for LOD0 with projected 2D/1D floorplan geometry (right, $d_2 = 1$) for the case of $d_1 = 0$, that is, the spatio-semantic CityGML target structure of building elements.

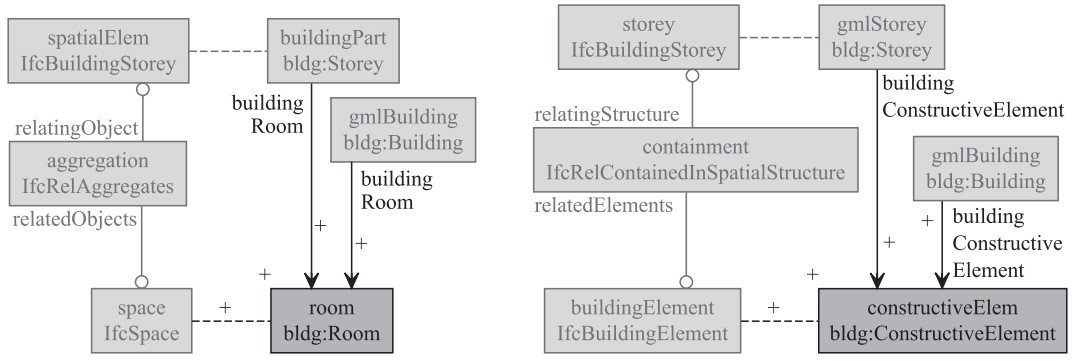
4.3 | Rule sets for spatio-semantic mapping

We now construct triple graph transformation rules from the graph structures outlined in Section 4.1. For each rule, we select one source and one target graph structure as templates and add connecting graph edges between related source (IFC) and target (CityGML) nodes. Ideally we would obtain a modular rule set structure such that we have a rule or a set of rules for each option in each dimension and these subsets can be freely combined throughout the dimensions as per the choice of the dimensions' options. However, we realized that this is not possible. As can be seen from the source and target graphs in Section 4.2, some source and target structures are specific



(a) Attachment to general structure (b) Geometry LOD and projection

FIGURE 6 Spatio-semantic CityGML target templates for constructive elements



(a) Rule 1.4: room nodes (b) Rule 2.1: constructive element nodes

FIGURE 7 Rules to create (concrete subtypes of) abstract space nodes in the CityGML graph and attach them to the existing triple graph instance

to options in more than one dimension. As a result, we end up with two sublayers of rules where the rules in each sublayer are marked with some dimension's choice and for each of the eight variants we select from these groups accordingly. The rules in the first sublayer pertain to the choice of d_1 and the rules in the second sublayer pertain to the combined choice of d_0, d_1 and d_2.

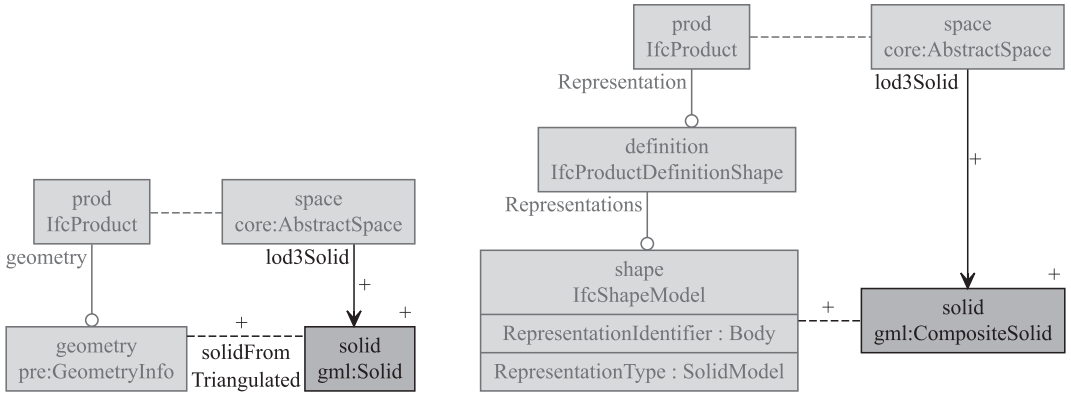
In the following, we focus on rules for the variants with building elements ($d_1 = 0$) and unprojected 3D (LOD3/4) geometry ($d_2 = 0$). We have described the construction of other rules (with $d_1 = 1$ for semantic surfaces and $d_2 = 1$ for floorplans) in more detail before (Konde, Tauscher, Biljecki, & Crawford, 2018; Tauscher & Stouffs, 2019).

Technically, spaces and rooms belong to the hierarchical spatial structure handled in the general structure layer (Section 3). From a spatio-semantic point of view, spaces and rooms are similar to building elements or constructive elements, both on the IFC and CityGML side. This can be seen from Figures 5a and 6a.

By combining the corresponding source and target templates from Figures 5a and 6a. we get the two rules shown in Figure 7 for the creation of CityGML rooms from IFC spaces (Figure 7a) and of CityGML constructive elements from IFC building elements (Figure 7b).

The first sublayer contains the rules that create the main spatio-semantic units in CityGML as per choice of dimension d_1: building elements versus boundary surfaces.

Rule 2.1 (Figure 7b) is included in the rule set for $d_1 = 0$. For the choice of $d_1 = 1$ we have a set of rules to create semantic surfaces in CityGML from the space boundaries in IFC. The semantic surface type in CityGML



(a) Rule 3.1, variant 1 (triangulated geometry)

(b) Rule 3.2, variant 2 (original IFC geometry)

FIGURE 8 Example rules to access IFC geometry nodes and create GML geometry nodes in the CityGML model

is determined from a combination of the building element type (e.g., wall) and the type of bordering space (e.g., interior).

Depending on the d_1 choice, the connected pair of an IFC and a CityGML node we end up with will be of one of two types. For $d_1 = 0$ (rules in Figure 7), we have a pair of IfcProduct (supertype of both `IfcSpace` and IfcBuildingElement) and core:AbstractSpace (supertype of both `bldg:Room` and bldg:ConstructiveElement). For $d_1 = 1$ (examples omitted for brevity), we have a pair of IfcRelSpaceBoundary and con:AbstractConstructionSurface (supertype of all semantic surfaces).

The rules in the second sublayer assign the actual geometry to the spatio-semantic units created with rules from the first layer.

These rules combine an IFC source template (e.g., one of Figure 5b according to the choice of d_0 with a CityGML target template (Figure 6b) according to the choice of d_2). In addition, we have two different types of exit node pairs from the first sublayer to attach the geometry to, so that we will have a basic set of eight rules in this layer, one for each variant. Figure 8 shows two of them.

After creating the GML geometry nodes, these have to be populated with lower-level geometry elements down to the actual coordinates. This is achieved with another layer of rules, described in the next section.

5 | GEOMETRY LAYER

In the literature there are two fundamentally different ways to handle geometry during the conversion process. Some proposed solutions delegate the geometry processing to a dedicated framework or tool. For example, the solution of Jusuf, Mousseau, Godfroid, and Soh (2017) is based on FME. Others access and process the original IFC geometry directly, such as Khalili and Chua (2015) and Zhu, Wang, Wang, Wu, and Kim (2019).

We have tried both approaches and these attempts are reflected in option d_0 in the spatio-semantic layer. The floorplan rule sets described in Konde et al. (2018) use the preprocessing approach for projected solids (variant 5 in Figure 4) and the original geometry approach for projected vertical surfaces or axes (variant 8). The rule set used for evaluation in Section 7 applies a combination of these submodules. The remainder of this section describes these two approaches.

5.1 | Preprocessed triangulated geometry

For the preprocessing of the geometry we use IFCOpenShell (http://ifcopenshell.org/), integrated in BIMserver as a render plug-in. IFCOpenShell resolves the IFC geometry in 3D model context to triangulated surfaces with coordinates in a global coordinate system for all products. The triangulation is carried out upon storing IFC data in BIMserver. The resulting triangulated geometry is stored in BIMserver mainly for visualization purposes.

This geometry assigned to a particular product can be retrieved from the server and accessed as shown earlier in Figure 5b (left graph). The resulting GeometryInfo node follows a typical data structure for triangulated irregular networks . An indexed list of nodes with coordinates is accompanied by a list of triangle meshes represented as triples of indices.

The conversion of such a preprocessed data structure into GML surfaces and triangle patches is straightforward. We execute this mapping in a procedural way in a so-called "converter". Converters are implemented as functions in the conversion engine and triggered during application of a rule that specifies the use of the converter. This particular converter is called "solidFromTriangulated" and invoked during every match and application of rule 3.1 (Figure 8a); the function expects a preprocessed geometry entity as input and generates the corresponding GML solid as its result.

It would be possible to cover this mapping with graph transformation rules as well, mapping the preprocessor data structure to the GML data structure.

5.2 | Original geometry

Geometry conversion based on the original IFC geometry is an alternative approach that does not rely on any preprocessing. We have only implemented such rules for selected IFC geometry types, so as to complement the preprocessed geometry rules in our conversion where necessary. In particular, this is necessary for geometry that is not handled by the preprocessor.

For example, some geometry types[4] are already implemented in Revit export, but not yet in IfcOpenShell. As another example, two-dimensional IFC geometry (e.g., storey footprints or a surface boundary geometry) is not accessible through the BIMserver render engine plug-in.

When combining preprocessed and original IFC geometry rules, we have to make sure that the corresponding spatio-semantic rules are applied properly. Figure 9 shows how rule 3.2 is extended to only match for representation items of type IfcPolygonalFaceSet.

5.2.1 | Mapping of geometry decomposition

Starting from a pair of an IFC entity of type IfcShapeModel and a GML object of type CompositeSolid as produced during application of rule 3.2 or derivatives (Figure 9), the following rules proceed to the decomposition of geometries. For the example case of IfcPolygonalFaceSet such rules are depicted in Figure 10. Ultimately, the conversion invokes a converter to resolve and convert the coordinates, here called "index2poly", and converts an IFC entity of type IfcIndexedPolygonalFace into a GML object of type Polygon. Other geometry types are deconstructed on the IFC side and recomposed on the CityGML side in a similar way.

The general mapping of solids in IFC to solids in CityGML seems quite straightforward, but we realized there was a mismatch in how solid conditions are defined in IFC and in CityGML. There is an adequate mapping as just shown, but the CityGML application schema contains geometric and topological restrictions that are not imposed on the IFC side. Thus, with this mapping, the CityGML constraints cannot be guaranteed and valid IFC input may

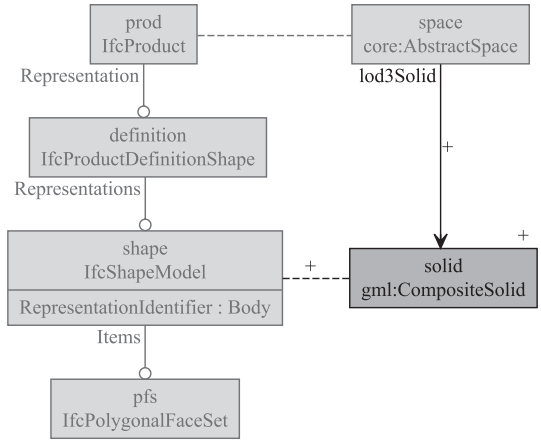


FIGURE 9 Restriction of rule 3.2 (Figure 8b) to a particular geometry type

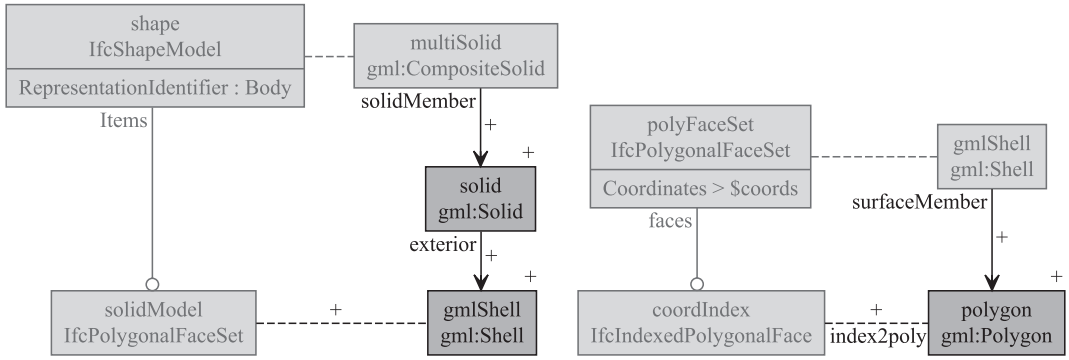


FIGURE 10 Exemplary rules to convert original IFC geometry to CityGML geometry

convert to erroneous CityGML output, containing, for example, incorrect disconnected and overlapping geometries. Additional validation before rule application could be a remedy.

5.2.2 | Positioning

In order for the converter (e.g., "index2poly") to calculate the coordinates in the global coordinate system, it needs to take into account the recursive relative positioning of products in IFC. In our rules, we obtain the position from previously processed nodes using context nodes. Figure 11 shows how the placement is stored in the context for an IFC building element. Subsequent rule applications and invoked converters can then access the current placement from the context.

If the placement is given as a local placement relative to another product's placement, the placements have to be resolved recursively. The resolution of relative object placement is described multiple times in the literature, but not always correctly. For instance, Zhu et al. (2019) assume the reference direction (x direction) to be given perpendicular to the z-axis, which is not necessarily the case.

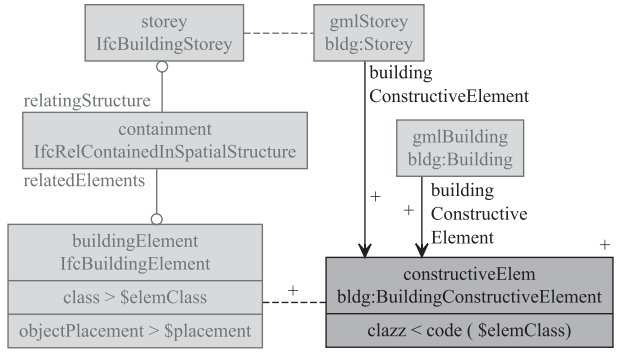


FIGURE 11 Rule 2.1 (Figure 7b) with getters and setters; placement getter supports geometry conversion

As an authoritative reference, the EXPRESS file (buildingSMART, 2016) contains the definition of a function (Listing 3) with two vectors as parameters, such as the attributes of IfcAxis2Placement3D, and derives the three pairwise orthogonal axes for the local coordinate system. The function call `IfcFirstProjAxis(D1, RefDirection)` projects the reference direction into the plane perpendicular to the z-axis `D1`, which is only necessary because `RefDirection` may not be orthogonal to the z-axis in the first place.

```
1   : LIST [3:3] OF IfcDirection;
2   LOCAL
3     D1, D2 : IfcDirection;
4   END_LOCAL;
5   D1 := NVL(IfcNormalise(Axis), IfcRepresentationItem()
6             || IfcGeometricRepresentationItem ()
7             || IfcDirection([0.0,0.0,1.0]));
8   D2 := IfcFirstProjAxis(D1, RefDirection);
9   RETURN ([D2,
10    IfcNormalise(IfcCrossProduct(D1,D2))\IfcVector.Orientation,
11    D1]);
12 END_FUNCTION;
```

LISTING 3. EXPRESS function IfcBuildAxes

5.2.3 | Normal definition for extrusions

The definition of normals for an extruded area solid is another issue where IFC and CityGML are not intuitively compatible. In IFC, the profile definition does not have an orientation, because after all it is a curve and not a surface. Consequently, the IFC specification does not assign any meaning to the order of the points of a closed polycurve.

In GML, however, every curve or surface has a default orientation and point order is meaningful. Thus to derive the faces of extrusion during conversion, we need to guarantee the orientation and enumerate the points counterclockwise when looking at the face from the direction opposite to its normal, that is, onto the positive face side.

Here, we describe one way to figure out the correct order of the polycurve points. We first determine the order of profile points for the bottom face of the extrusion, which will face away from the extrusion. We assume that the polycurve points define the face counterclockwise. We then check whether the normal points in the same way as the extrusion vector, and if it does we invert the order of the polycurve points.

To obtain the normal, we check the sign of the area A of the profile. The profile is defined in two dimensions in the xy-plane of the local coordinate system:

$$2A = \sum_{i=1}^{r-1} (x_{i+1} - x_i)(y_{i+1} + y_i) \tag{1}$$

If $2A$ is positive we have a profile normal $N_P = (0, 0, 1)$, otherwise it is $N_P = (0, 0, -1)$. If the angle between N_P and the extrusion vector E is less than $90°$ (i.e., $N_P \cdot E > 0$) then the two vectors point in the same direction and we have to reverse the polycurve points. Note that an exact $90°$ angle should not occur, since it would create a degenerate extrusion, where the profile is extruded along a direction in its host plane.

Unlike Zhu et al. (2019), we apply the extrusion in the local coordinate system and apply the transformation to absolute coordinates only afterwards.

6 | PROPERTIES LAYER

The property layer[5] comprises rules to process non-spatial attribute values (information that is not related to either the semantic structure or geometry) from the IFC graph, transform these values and add the results to the CityGML graph. These added properties supplement the semantic structure and geometry and provide additional non-geometric information for the various use cases the CityGML file might be used for.

Property rules assume that the semantic structure and geometry of the graph are already present. During the application of a property rule specific values are extracted from and added as attributes to matching nodes of the IFC and CityGML object graphs.

6.1 | Property graph templates

The structure of the property rules may differ depending on where and how the relevant properties are to be found in the IFC graph as well as where and how they should be inserted in the CityGML graph. In order to identify common patterns that could be used to guide property rule creation, we first explored how properties and attributes appear in IFC and CityGML, respectively.

Both on the IFC and the CityGML side we find attributes in two flavours: they are either direct attributes of semantic entities or property entities in their own right. Accordingly, they appear as direct key–value pairs on a node of the instance graph or as a node itself with multiple attributes. Figure 12 shows both options.

6.2 | Types of property rules

The different possible configurations used to store information in IFC and CityGML, as described in Section 6.1, can be combined in a number of ways and result in the following possible rule structures:

1. from a direct attribute in IFC to a direct attribute in CityGML;
2. from a direct attribute in IFC to a new node in CityGML;
3. from a property in IFC to a direct attribute in CityGML; and
4. from a property in IFC to a new node in CityGML.

In Section 6.2.5 we consider how many sources contribute to a target.

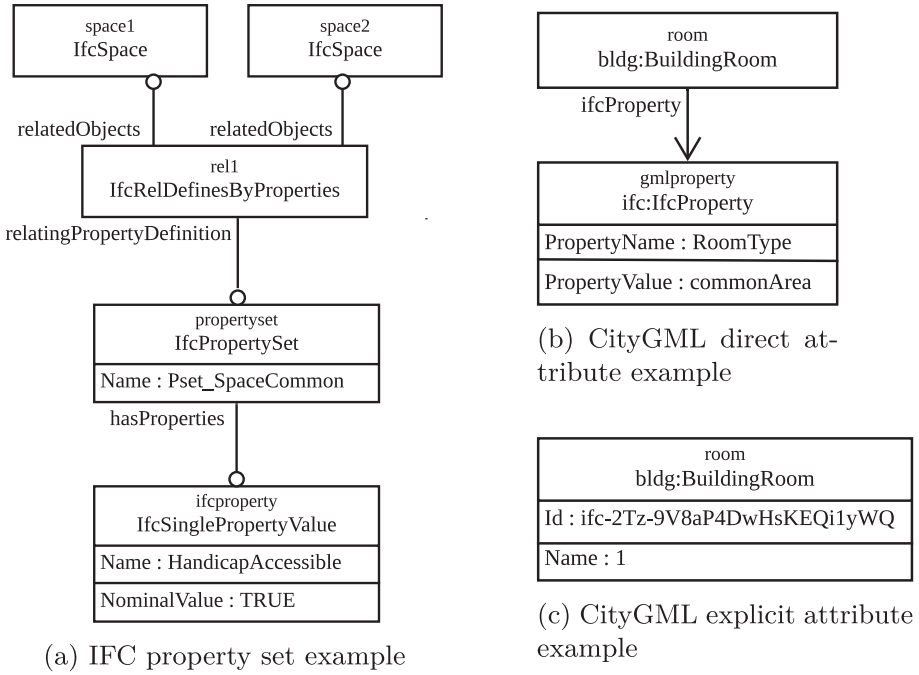


FIGURE 12 Template graph examples for attributes and properties in: (a) IFC; and (b, c) CityGML. In the case of (a), one property node is shared by multiple referencing nodes: Multiple `IfcSpace` objects refer to the same HandicapAccessible property

6.2.1 | Direct attribute to direct attribute

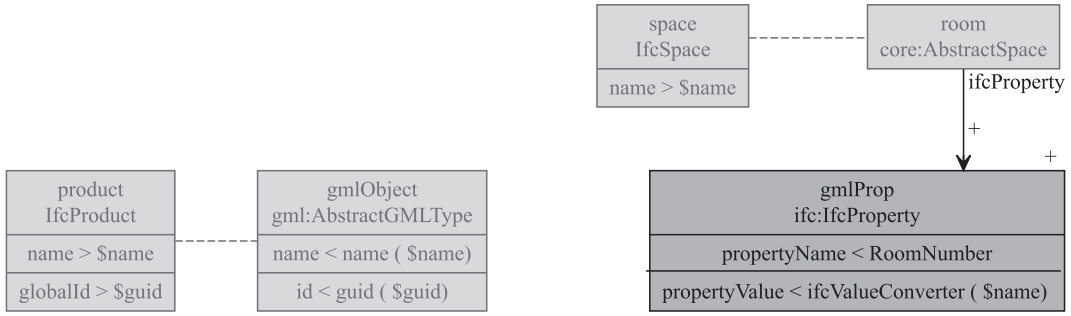
Such rules extract a value directly from the entry IFC node and add an attribute directly to the specified entry CityGML node. One example is the transfer of IDs or names from the IFC file to the CityGML file (Figure 13a). Functions `name` and `guid` convert the values to fulfil constraints of the target schema.

6.2.2 | Direct attribute to new node

Such rules extract a value directly from an existing IFC node, create a new CityGML node and add it to the CityGML graph by creating an edge between an existing CityGML node corresponding to the initial IFC node and the newly created CityGML node. One example is shown in Figure 13b. This rule applies to an IFC node of type `IfcSpace` and an existing corresponding CityGML node of type `AbstractSpace`. The `name` attribute, which many BIM software tools populate with the room's number during export, is extracted and inserted into the CityGML graph as a newly created `IfcProperty` node.

6.2.3 | Property to direct attribute

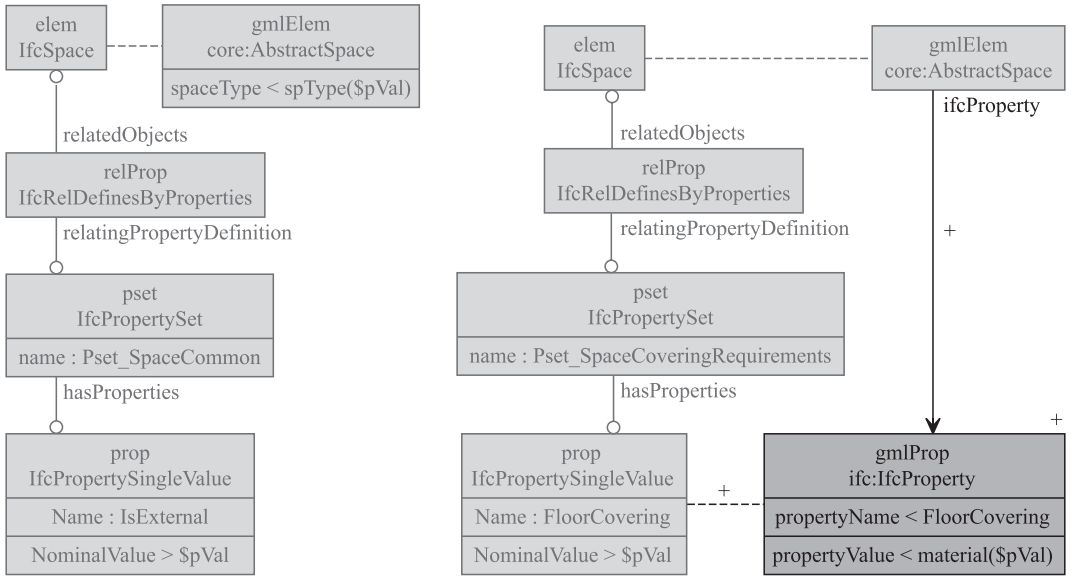
Such a rule extracts a value from a property or quantity that is located down a path from the entry IFC node and adds an attribute directly to the correlated existing CityGML node. One example is shown in Figure 14a. This rule extracts the `IsExternal` property from the Pset_SpaceCommon property set of an `IfcSpace` typed entity



(a) Rule 5.1: converts global ID, name

(b) Rule 5.2: converts room number

FIGURE 13 Example of rules to convert direct IFC attributes to: (a) CityGML attributes; or (b) CityGML property nodes



(a) Rule 5.3: converts isExternal property

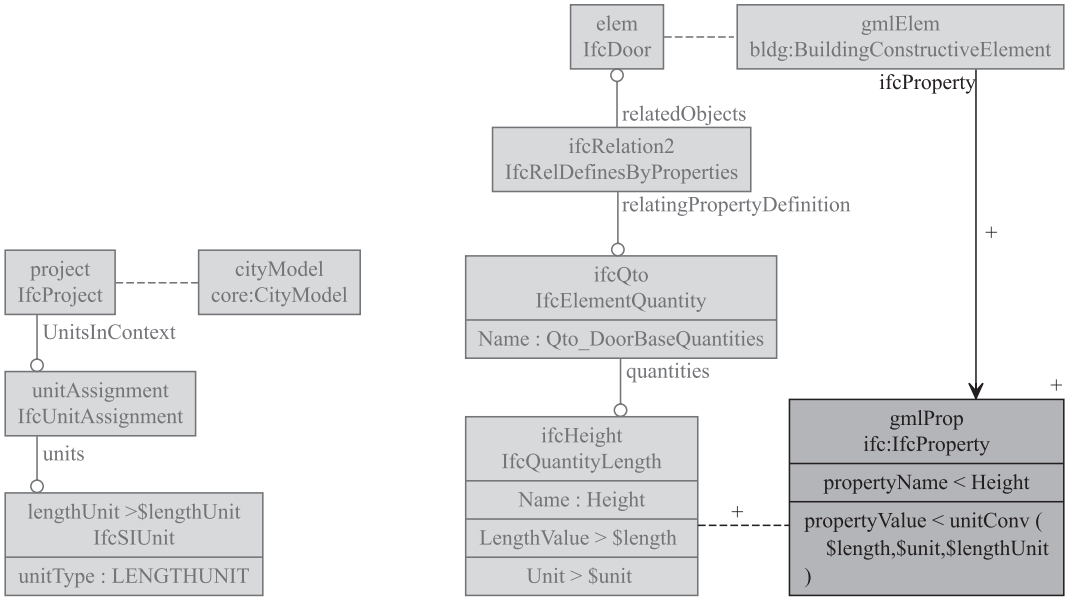
(b) Rule 5.4: converts floor material

FIGURE 14 Example rules to convert IFC property nodes to: (a) CityGML direct attributes; or (b) CityGML property nodes

and uses the value to determine the room's `spaceType`, assigning it to the respective `AbstractSpace` type `gmlElem` node in the CityGML graph.

6.2.4 | Property to new node

Such a rule extracts a value from a property or quantity that is located down a path from the entry IFC node, creates a new CityGML node and adds it to the CityGML graph by creating an edge between an existing CityGML node corresponding to the initial IFC node and the newly created CityGML node. Furthermore, it creates a correlation between the property nodes in IFC and CityGML. One example is the FloorSurfaceMaterial rule,



(a) Rule 5.5: extracts default length unit from IfcProject

(b) Rule 5.6: extracts height value and specific length unit from IfcDoor

FIGURE 15 Example of case where two rules are required to convert one property

shown in Figure 14b. This rule extracts the value of the IFC property with name `FloorCovering` from the Pset_SpaceCoveringRequirements property set of an IFC node of type `IfcSpace` and sets it as a new `IfcProperty` node of the corresponding existing CityGML node.

6.2.5 | Multiple sources to single target

Some rules require multiple attributes and properties from the IFC graph to be combined in order to determine the desired value to be inserted into the CityGML graph.

If these attributes and properties are specified in a location of the IFC graph far away from each other, it may be useful to store one of the values in context during the application of another rule that traverses the same node. One example of such a case is door height. In IFC, units can be omitted in measurement properties. In such cases, the properties take on the default units, defined at the level of the `IfcProject` node, which is processed much earlier. To transfer such a property would require two rules: one rule to extract the units from `IfcProject` and store them in context (Figure 15a); and one rule to extract that height value from the door, adjust it using the stored units, and assign it to a new node in the CityGML graph (Figure 15b).

One issue with such rule construction is that the order in which the rules are applied is crucial. In this case, the first rule has to be applied before the second rule in order to work.

7 | RULE SET APPLICATION TO SAMPLE PROJECTS

Here we present the results from applying a selected rule set from the layers described in Sections 3–6 to five sample projects and IFC data sets:

(A) two-storey residential building (synthetic data set);
(B) Revit advanced tutorial model (IFC4 RV export);
(C) ten-storey office building (BCA Academy);
(D) anonymized estate 1; and
(E) anonymized estate 2.

The rule set employed consists exactly[6] of the rules described as representatives of the layers in Sections 3–6. Additionally, the geometry layer contained rules for extruded geometry with indexed polygons similar to rules 4.1 and 4.2 as well as further rules in the property layer similar to rules 5.1–5.4.

A visual impression of the conversion results is given in Figure 16. The images were created with FZKViewer 5.0 (build 972).

For each project, we evaluated the applicability of each single rule, that is, how many times a rule could be applied to a particular project. These numbers are collected in Tables 2–4 for each of the representative rules from Sections 3–6.

We also measured and recorded the performance of the rule set implementation for each data set, using two machine set-ups, as seen in Table 5. The results are shown in Table 6.

Finally, we assessed the presence of particular configurations of elements in the CityGML files generated, and compared them with expectations derived from the intended use cases and IFC source data. We also verified geometric integrity of the results, tracing any inconsistencies back to the input data, the rule set or the converter source code.

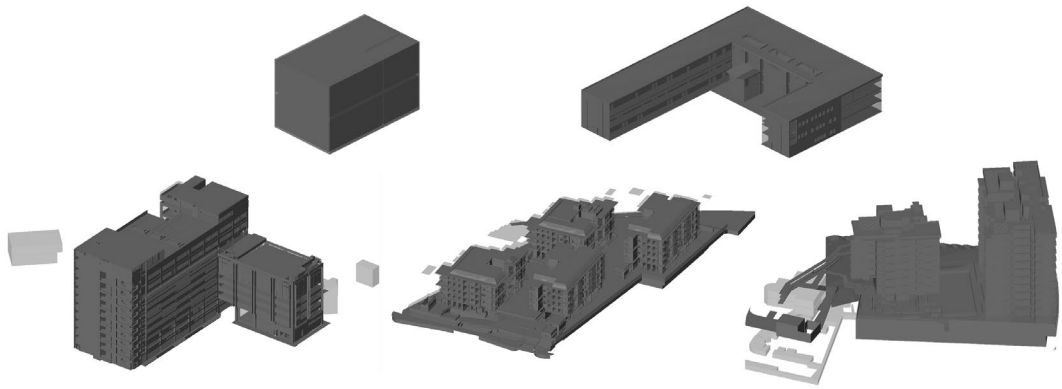


FIGURE 16 Visualization of resulting CityGML files for the two-storey residential building data set (top left), Revit advanced tutorial model data set (top right), BCA Academy data set (bottom left), Estate D data set (bottom centre) and Estate E data set (bottom right)

TABLE 2 General structure rule applicability, projects A–C

		No. of applications		
Rule no.	Rule name	Project A	Project B	Project C
1.0	Project2CityModel	1	1	1
1.1	Building2Building	1	0	0
1.2	SiteBuilding2Building	0	1	1
1.3	Storey2Storey	3	5	16
1.4	Space2Room	6	91	709

TABLE 3 Spatio-semantic rule applicability, projects A–C

Rule no.	Rule name	No. of applications		
		Project A	Project B	Project C
2.1	building element	15	460	11,634
3.1	3D building elements preprocessed (variant 000/1)	27	5,411	21,970
3.2	3D building elements original (variant 001/2)	24	5,355	21,648

TABLE 4 Property rule applicability, projects C–E

Rule no.	Rule name	Rule type	No. of applications		
			Project C	Project D	Project E
5.1	IfcGuid2GmlId + GmlName	1	17,153	98,283	13,560
5.2	RoomNumber	2	701	2,247	462
5.3	IsExternal2SpaceType	3	701	2,247	0
5.4	FloorSurfaceMaterial	4	56	0	0
5.6	DoorHeight	4	571	1,570	0

TABLE 5 Machines and BIMserver set-ups used to test each data set

Projects	Machine	Server
A, B, C, D	Intel(R) Core(TM) i7-7700 CPU 3.60 GHz 4 cores, 8 logical processors 16 GB RAM	Local BIMserver Max. heap size: 8 GB
E	Intel(R) Core(TM) i7-7920HQ CPU 3.10 GHz 2 cores, 2 logical processors 32 GB RAM	Shared BIMserver Max. heap size: 10 GB

TABLE 6 Runtime performance metrics for each data set

Model	IFC size (kB)	CityGML size (kB)	Fetch time (ms)	Conversion time (min)	Peak memory usage (MB)
A	33	121	252	0.05	750
B	20,872	124,606	3624	0.04	2,200
C	80,785	187,706	21657	9.67	2,500
D	127,262	142,921	42767	9.21	3,000
E	90,657	730,098	15369	5.34	7,000

With visualization being a popular basic usage of the data generated, we generated visualizations and conducted manual plausibility checks for the presence of expected elements and attribute values as well as geometric integrity. We further used FME and val3dity (Ledoux, 2018) for analysis and checking of the results.

8 | DISCUSSION

In this section we discuss our results—both the developed rule set modules described in Sections 3–6 and the application to five sample data sets presented in Section 7.

8.1 | Practicability of the graph transformation approach

We found the approach versatile enough to cover the whole conversion process and graph transformation rules sufficiently expressive to specify IFC-to-CityGML conversion in large part. Since we considered the option to resort to procedurally implemented converters not only for primitive data, but also for more complex structures, we cannot claim that full coverage is possible, in particular for rules in the geometry layer, although the results are promising. We have only superficially tapped the potential for formal verification, counting rule applicability numbers and coverage percentages.

With regard to the assumed advantages of representing the conversion as graph transformation rules, we found that the visual nature of the rules indeed supported understanding of the conversion logic and fostered communication between persons with different levels of capability in terms of computer programming and software development. The empirical evaluation of how graph transformations are indeed more approachable for domain engineers will be interesting but exceeded the scope of the project.

A limitation of expressiveness that we have not solved fully but only with workarounds concerns rules for unbounded recursive structures such as the IFC spatial structure or placements in IFC. This can be worked around by only creating rules for some defined combinations or by using context.

In addition to the issues reported here, we faced some limitations that are not induced by the graph transformation approach as such or the rule set, but trace back to the conversion engine and the implementation priorities and decisions we made there.

8.2 | Performance

The conversion times are within acceptable limits for current hardware and usual IFC file sizes and for the use case of asynchronous batch conversion ahead of further CityGML processing. They are too high for synchronous conversion and use of converted data, as required for real-time applications. The conversion algorithms were not optimized in any way for performance.

The jump in conversion time between projects A/B and C/D is likely due to full utilization of the memory assigned to the JVM (Java Virtual Machine) running the conversion client. Projects C and D are of a similar order of magnitude, with D having larger input but a slightly smaller portion converted. The lower conversion time for project E is due to the larger memory available.

File sizes of CityGML increase compared to IFC input by a factor of up to 8. This increase can be attributed to the overhead of the XML format as compared to the STEP physical file format of the IFC input. The XML format contains attribute names, often even twice in opening and closing tags, and has possibly deep indentation levels. Another reason for the increase may be the resolution of prototype geometry and reused non-geometric attributes in IFC into attributes of individual features in CityGML. With reused geometry and property sets occurring with different frequency in various IFC input, this may also explain the nonlinear increase in file size.

8.3 | Modularity

Capturing the conversion details in rules proved beneficial for reuse of partial conversions across varying input and use cases. Beyond modularity on the rule level, we aimed to achieve modularity at a larger and smaller level:

for groups of rules and for parts of rules. Coarse-grained modularity with independent rules for certain options has been achieved exemplarily for the spatio-semantic layer. We have shown one of the options in detail in this article. In order to use the suggested modularity concept efficiently in practice, this would require metadata for the preconfigured rule set modules and some form of management for the dependencies between rules and groups of rules.

The management of a large set of partially similar rules has proven to be a major difficulty in rule development, in particular for rules in the property layer. This can be tackled with fine-grained modularity below the scale of a single rule. One possible approach is to parameterize the rules and generate the final rules from a list of parameters. This approach is confined to very similar rules, which only differ in one or two parameters, such as rules 5.4 and 5.6. The categorization of rules into types according to their formal appearance (as done for the property layer) is a first step towards this kind of fine-grained modularization. A more general take on this is rule refinement (Anjorin, Saller, Lochau, & Schürr, 2014). This method allows for the reuse of rule parts by deriving refined rules from base rules through addition and overriding of elements. As an example of where rule refinement would be useful, consider rules 1.1 and 1.2 with the bigger part of nodes and edges identical in both rules.

8.4 | Requirements on source and target models

As a direct consequence of the adaptable conversion approach with different rule sets covering different portions of the IFC and CityGML schemas, the choice of a rule set also has implications on the source and target sides. A rule set implicitly defines its scope and thus requirements of the IFC input to be fulfilled for successful application and, likewise, specific capabilities of what kinds of CityGML output can be generated or not. These inherent requirements and capabilities can be extracted from the rule sets and used to check a rule set against a particular given IFC input or desired CityGML output for a particular use case.

Non-fulfilment of the requirements of a rule set will lead to missing results in the conversion. We were not concerned with heuristic methods to fill in such potential gaps or automatically select or construct matching rules. Our rule sets do rely as much as possible on data being present in IFC, where the IFC standard provides certain constructions. Preprocessing to populate, enhance or transform the IFC to meet these requirements should not be mixed with the conversion to separate concerns.

For example, on the source side, the rule set using preprocessed geometry relies on triangulated geometry being present in an external geometry model. Likewise we could have a rule set that relies on triangulated face sets being present in the IFC as original geometry. Or we may prefer polygonal tessellation as input, even though, in practice, preprocessing engines such as IfcOpenShell default to triangulation. The resolution of Boolean operations in constructive solid geometry representations is another example of preprocessing that may or may not be required.

The spatio-semantic rules also expect spaces with geometry to be existent in the IFC. A similar case are the semantic surface variants in the spatio-semantic layer. These rely on space boundaries with connection geometry being present in the IFC input. such implications similarly exist on the CityGML side. For example, the spatio-semantic mapping rules for building elements and LOD0 are only available in CityGML 3.0 (Kutzner & Kolbe, 2018) and cannot produce useful results for target environments where only CityGML 2.0 is supported. Some rules in the property layer rely on a dedicated application domain extension to hold the converted attributes and properties.

8.5 | Schema mismatches encountered

During development of the rules, we found several situations where the two schemas, IFC and CityGML, do not match in a trivial way. We summarize them here as a reference for potential pitfalls in conversion.

1. The granularity and flexibility of the spatial structure are different between IFC and CityGML. In particular, recursive structures with arbitrary depth on the IFC side are hard to map to the restricted structure on the CityGML side.
2. The reuse of resources (e.g., properties, geometry) is treated differently in EXPRESS with its network structure and CityGML (XML) with its tree-like structure.
3. The main spatial paradigm is a different geometry paradigm, IFC is mainly built around solid geometry, while CityGML is built around surface geometry.
4. The definition of aggregated solids varies.
5. Surface orientation varies.

9 | CONCLUSIONS

We have shown how graph transformation as a formal method can be leveraged to solve practical data integration problems such as conversion from IFC to CityGML adjusted to input model and use case requirements. To this end we developed a modular framework for IFC-to-CityGML transformation rules, an exemplary rule set and demonstrated its application to a range of input data.

9.1 | Potential of the approach

This approach has the potential to be applied to other domains. It could be used similarly for converting IFC to other formats such as IndoorGML or Green Building XML (gbXML) or for deriving CityGML from additional sources such as sensor data or public transportation information.

Domain specialist involvement in developing such conversions is facilitated because conversion details are explicitly represented in graphical form or with a DSL instead of implicitly woven into a conversion algorithm written in general programming language. Yet, the specification is formal and allows for quantitative analysis of conversion specifications and executions.

Moreover, the conversion specification is broken into reusable pieces and thus naturally allows for configurable solutions by simply reassembling the pieces. As such it is particularly useful for domains with heterogeneous input and use case structures and for application scenarios with unstable data models.

The full potential can be unfolded when the correlation between domain models is specified as a TGG. Operational transformation systems, such as the conversion presented here, can then be derived from the general grammar.

9.2 | Future work

This research has thrown up some questions in need of further investigation. For example, it would be interesting to empirically evaluate the usability of the graph transformation approach with domain specialists.

It is suggested that more of the procedurally specified converters are turned into declarative specifications. Further research is needed to understand how far this approach can be taken, in particular for geometry. A greater focus on geometry is recommend also to ensure validity of CityGML results, for example by extending some rules with additional constraints on the IFC geometry. For practical application and management of larger rule sets, implementation of the suggested modularity concepts beyond the level of a single rule will need considerably more work—for example, rule refinement for fine-grained modularity and meta data and dependency management for coarse-grained modularity.

Although this article has not focused on the conversion algorithms as such, natural progressions of this work will include further development of the actual rule application. Additional tracking of converted/traversed nodes and coverage/completeness measures of the conversion can also provide useful insight for the developers of a

rule set. Investigation of the actual conversion input and output on a larger sample, and quantitative investigation of IFC and CityGML characteristics other than file size (e.g., number of entities and references), as well as specifics with regard to particular rule sets, will allow more accurate conclusions on efficiency of the conversion. Conversion algorithms can also be evolved with performance optimization (e.g., by parallelization of rule application).

As an alternative to a hand-crafted solution, the use of an existing fully functional graph transformation library or framework could be a viable option to also align this research with current findings and advancements of the general graph transformation community.

ACKNOWLEDGMENTS

This material is based on research/work supported by the National Research Foundation under Virtual Singapore Award No. NRF2015VSG-AA3DCM001-008.

ORCID

Helga Tauscher https://orcid.org/0000-0003-0411-8770
Rudi Stouffs https://orcid.org/0000-0002-4200-5833

ENDNOTES

[1] A note on terminology is in order. In the graphs, we literally name the types from the schemas, for example `bldg:-ConstructiveElement` (for nodes: second line of the label). In the running text, however, for better readability, we embed the type in the natural language; for example, we speak of a ***constructive element*** instead of an ***object of type bldg:ConstructiveElement***.

[2] Opensource BIMserver: http://opensourcebim.org/bimserver, http://bimserver.org.

[3] This section is a revised and compressed version of part of Tauscher and Stouffs (2019).

[4] IfcPolygonalFaceSet, IfcExtrudedAreaSurface (with indexed polygon as extruded area), IfcCurveBoundedPlane (for horizontal connection surfaces), IfcSurfaceOfLinearExtrusion (for vertical connection surfaces).

[5] This section is based on Lim, Tauscher, and Biljecki (2019).

[6] The conversion was carried out with a draft version of CityGML 3.0, whereas this article contains updated diagrams according to the specification as released for public Request for Comments. The updates contain only attribute and class renames and as such do not affect the conversion.

[7] The attribute is named “clazz” here, because the conversion engine is implemented in Java where the attribute name “class” is already reserved.

REFERENCES

Anjorin, A., Leblebici, E., Kluge, R., Schürr, A., & Stevens, P. (2015). A systematic approach and guidelines to developing a triple graph grammar & In *Proceedings of the Fourth International Workshop on Bidirectional Transformations*, L’Aquila, Italy (pp. 81–95). CEUR-WS.org. http://ceur-ws.org/Vol-1396/p81-anjorin.pdf.

Anjorin, A., Saller, K., Lochau, M., & Schürr, A. (2014). Modularizing triple graph grammars using rule refinement. In S. Gnesi & A. Rensink (Eds.), *Fundamental approaches to software engineering* (pp. 340–354). Berlin, Germany: Springer.

Beetz, J., van Berlo, L., de Laat, R., & van de Helm, P. (2010). BIMserver.org: An open source IFC model server. In *Proceedings of the CIB-W78 27th International Conference on Applications of IT in the AEC Industry*, Cairo, Egypt (pp. 1–8). Ottawa, ON, Canada: CIB.

buildingSMART. (2016). *Industry foundation classes* (Technical Report Version 4, Addendum 2). Kings Langley, UK: buildingSMART International Ltd.

de Laat, R., & van Berlo, L. (2011). Integration of BIM and GIS: The development of the CityGML GeoBIM extension. In T. Kolbe, G. König, & C. Nagel (Eds.), *Advances in 3D geo-information sciences* (Lecture Notes in Geoinformation and Cartography, pp. 211–225). Berlin, Germany: Springer.

Deng, Y., Cheng, J. C., & Anumba, C. (2016). Mapping between BIM and 3D GIS in different levels of detail using schema mediation and instance comparison. *Automation in Construction*, *67*(1), 1–21.

Donkers, S., Ledoux, H., Zhao, J., & Stoter, J. (2016). Automatic conversion of IFC datasets to geometrically and semantically correct CityGML LOD3 buildings. *Transactions in GIS*, *20*(4), 547–569.

Ismail, A., Nahar, A., & Scherer, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard. In C. Koch, W. Tizani, & J. Ninic (Eds.), *Proceedings of the 24th International Workshop on Intelligent Computing in Engineering*, Nottingham, UK (pp. 146–157). Red Hook, NY: Curran Associates, Inc.

ISO. (2013). *Industry foundation classes (IFC) for data sharing in the construction and facility management industries* (Technical Report 16739). Geneva, Switzerland: International Organization for Standardization.

Jusuf, S., Mousseau, B., Godfroid, G., & Soh, J. (2017). Path to an integrated modelling between IFC and CityGML for neighborhood scale modelling. *Urban Science*, *1*(3), 25.

Khalili, A., & Chua, D. K. H. (2015). IFC-based graph data model for topological queries on building elements. *Journal of Computing in Civil Engineering*, *29*(3), 04014046.

Kindler, E., & Wagner, R. (2007). *Triple graph grammars: Concepts, extensions, implementations, and application scenarios* (Technical Report tr-ri-07-284). Paderborn, Germany: Department of Computer Science, University of Paderborn.

Konde, A., Tauscher, H., Biljecki, F., & Crawford, J. (2018). Floor plans in CityGML. *Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(4/W6), 25–32.

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New functions open up new applications. *PGF—Journal of Photogrammetry, Remote Sensing & Geoinformation Science*, *88*(1), 43–61.

Kutzner, T., & Kolbe, T. H. (2018). CityGML 3.0: Sneak preview. In T. P. Kersten, E. Gülch, J. Schiewe, T. H. Kolbe, & U. Stilla (Eds.), *Photogrammetrie-Fernerkundung-Geoinformatik-Kartographie 2018* (Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., Vol. *27*, pp. 835–839). Munich, Germany: Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V.

Ledoux, H. (2018). val3dity: Validation of 3D primitives according to international standards. *Open Geospatial Data, Software and Standards*, *3*, 1.

Lim, J., Tauscher, H., & Biljecki, F. (2019). Graph transformation rules for IFC-to-CityGML attribute conversion. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(4/W8), 83–90.

Nguyen, S. H., Yao, Z., & Kolbe, T. H. (2017). Spatio-semantic comparison of large 3D city models in CityGML using a graph database. *Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(4/W5), 99–106.

Schürr, A. (1995). Specification of graph translators with triple graph grammars. In E. W. Mayr, G. Schmidt, & G. Tinhofer (Eds.), *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science* (pp. 151–163). Berlin, Germany: Springer.

Smith, G., & Ledbrook, P. (2014). *Grails in action*. Shelter Island, NY: Manning Publications.

Stouffs, R., Tauscher, H., & Biljecki, F. (2018). Achieving complete and near-lossless conversion from IFC to CityGML. *International Journal of GeoInformation*, *7*(9), 355.

Sun, J., Liu, Y.-S., Gao, G., & Han, X.-G. (2015). IFCCompressor: A content-based compression algorithm for optimizing industry foundation classes files. *Automation in Construction*, *50*, 1–15.

Tauscher, H. (2019). Creating and maintaining IFC-CityGML conversion rules. *Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(4/W8), 115–122.

Tauscher, E., Bargstädt, H.-J., & Smarsly, K. (2016). Generic BIM queries based on the IFC object model using graph theory. In *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, Osaka, Japan (pp. 905–912). ICCCBE2016 Organizing Committee.

Tauscher, H., & Crawford, J. (2018). Graph representations for querying, examination, and analysis of IFC data. In J. Karlshøj & R. Scherer (Eds.), *EWork and eBusiness in architecture, engineering and construction: Proceedings of the 12th European Conference on Product and Process Modelling*, Copenhagen, Denmark (pp. 421–428). London, UK: CRC Press.

Tauscher, H., & Stouffs, R. (2019). Extracting different spatio-semantic structures from IFC using a triple graph grammar. In T. Fukuda, M.A. Schnabel, & M.H. Haeusler (Eds.), *Intelligent & informed: Proceedings of the 24th International Conference on Computer-Aided Architectural Design Research in Asia*, Wellington, New Zealand (pp. 605–614). Hong Kong: The Association for Computer-Aided Architectural Design Research in Asia.

Varró, D. (2006). Model transformation by example. In O. Nierstrasz, J. Whittle, D. Harel, & G. Reggio (Eds.), *Model driven engineering languages and systems* (Lecture Notes in Computer Science, Vol. *4199*, pp. 410–424). Berlin, Germany: Springer.

Vilgertshofer, S., & Borrmann, A. (2017). Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models. *Advanced Engineering Informatics*, *33*, 502–515.

Yao, Z., & Kolbe, T. H. (2018). A new approach to model transformation using graph transformation system. In T. P. Kersten, E. Gülch, J. Schiewe, T. H. Kolbe, & U. Stilla (Eds.), *Photogrammetrie-Fernerkundung-Geoinformatik-Kartographie 2018* (Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., Vol. *27*, pp. 831–834). Munich, Germany: Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V.

Zhu, J., Wang, X., Wang, P., Wu, Z., & Kim, M. J. (2019). Integration of BIM and GIS: Geometry from IFC to shapefile using open-source technology. *Automation in Construction*, *102*, 105–119.

How to cite this article: Tauscher H, Lim J, Stouffs R. A modular graph transformation rule set for IFC-to-CityGML conversion. *Transactions in GIS*. 2021;25:261–290. https://doi.org/10.1111/tgis.12723

APPENDIX A

We present the selected rules that form a self-contained rule set in context and give a brief description of the converters to facilitate their usage and custom implementation of the conversion with these rules.

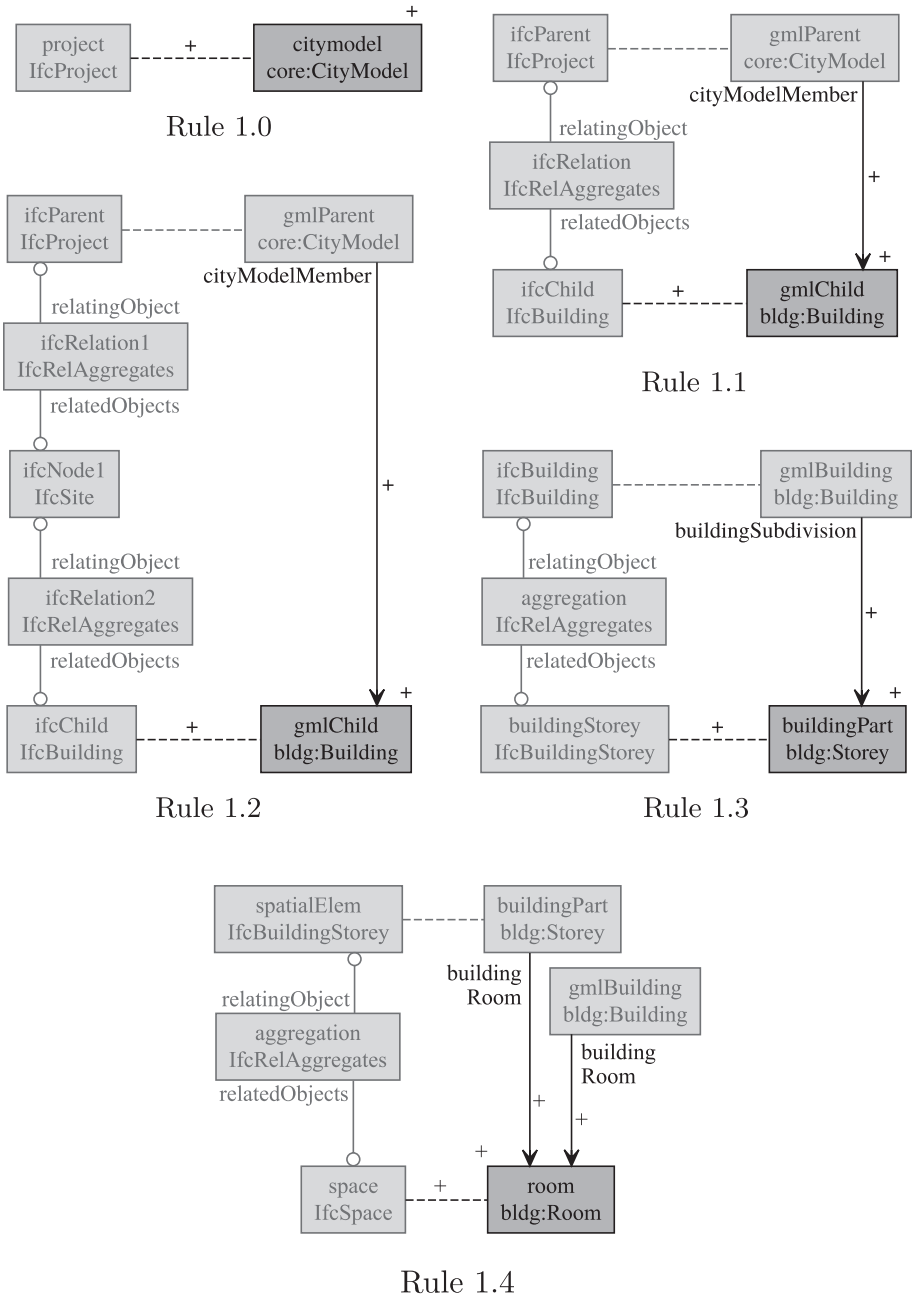


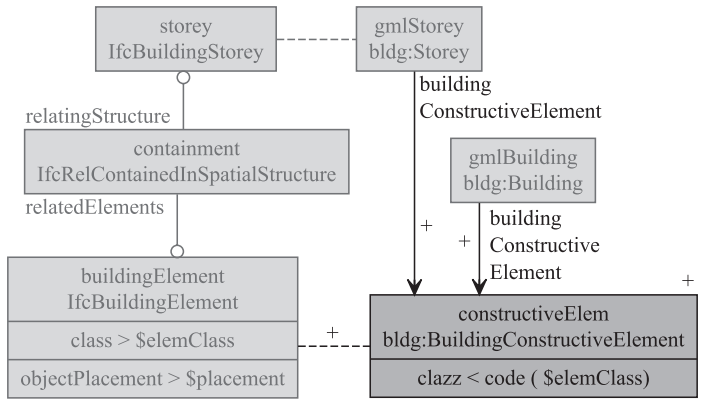
FIGURE A1

General structure layer

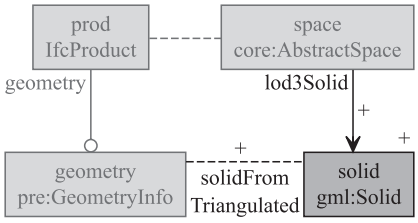
Spatio-semantic layer

code: Value converter to convert an IFC entity type (subtype of IfcBuildingElement) into a code point of the CityGML code list for the class attribute of BuildingConstructiveElement.[7]

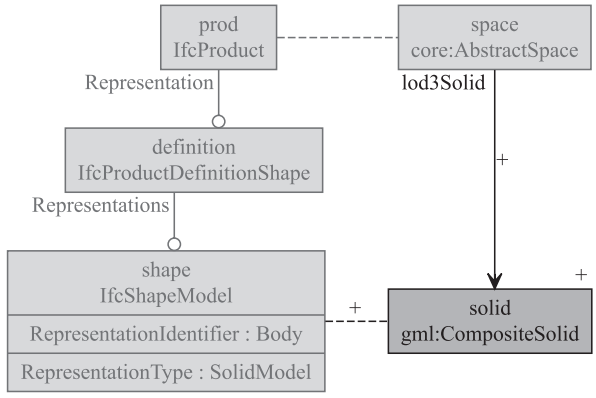
solidFromTriangulated: Node converter to convert preprocessed triangulated geometry from IfcOpenShell into a GML solid, including the object graph defining the solid.



Rule 2.1



Rule 3.1



Rule 3.2

FIGURE A2

Geometry layer

index2poly Node converter to convert original IFC geometry of type IfcIndexedPolygonalFace into a GML polygon. The converter accesses the actual coordinates by index. These coordinates are stored in a context variable "coords" from the IfcPolygonalFaceSet instance containing the respective face.

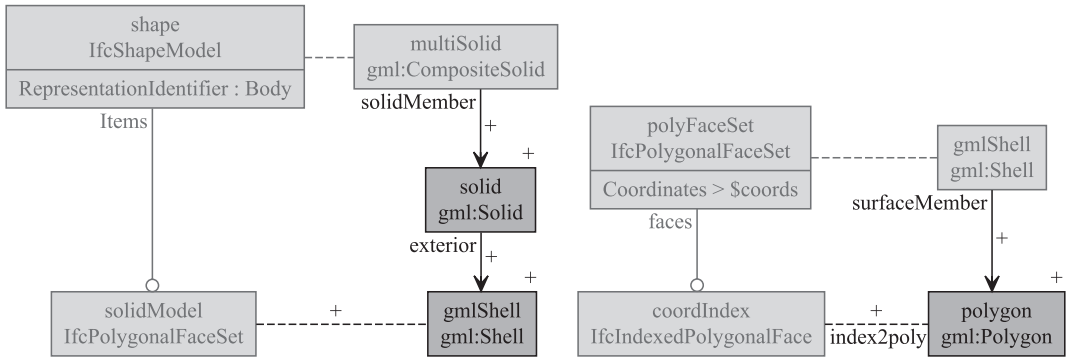


FIGURE A3

Properties layer

name Value converter to convert IFC name strings to GML name strings. Constraints vary between the two standards.

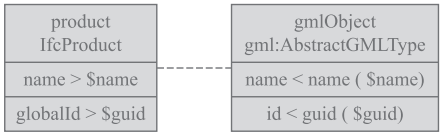
guid Value converter to convert IFC's globally unique IDs to GML GUIDs. Constraints and encoding vary between the two standards.

ifcValueConverter Value converter to convert IFC property value types to GML. The converter recognizes the various types of IFC property values and creates the respective GML values.

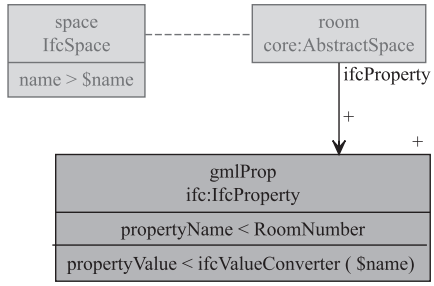
spType Value converter to convert the IFC IsExternal property value to the respective value of the CityGML spaceType enumeration.

material Value converter to convert an IFC material label (in this case for a space's floor covering) to a property value in the IFC application domain extension.

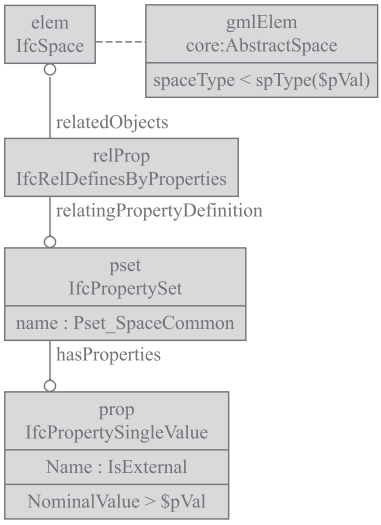
unitConv Value converter to convert a numeric IFC value (quantity) into the corresponding CityGML value taking default and specified IFC units and required CityGML units into account.



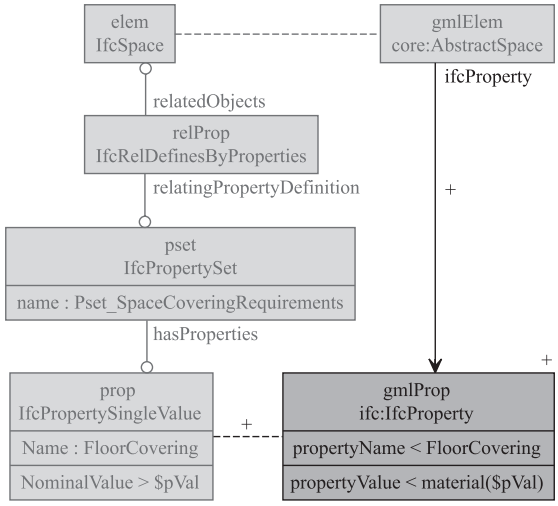
Rule 5.1



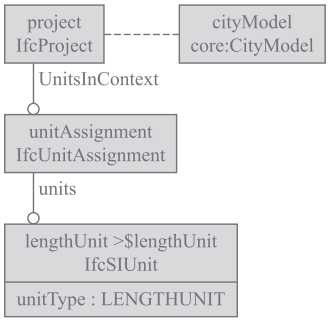
Rule 5.2



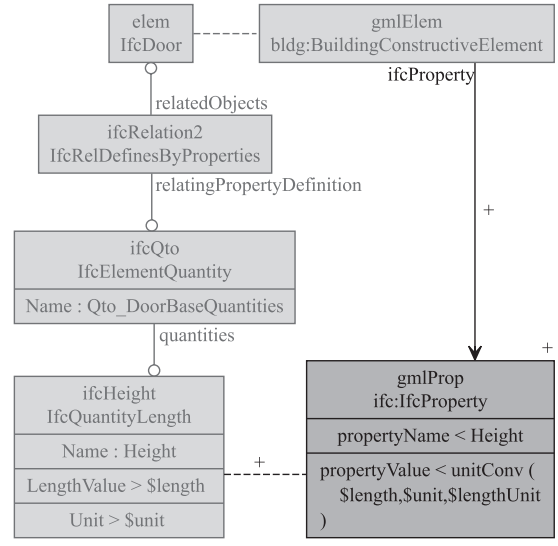
Rule 5.3



Rule 5.4



Rule 5.5



Rule 5.6

FIGURE A4