BauSIM 2024 in Wien
23. September – 26. September

Metamodels: Built-In BIM Compliance Checking (1)

Galina Paskaleva[1,2], **Thomas Bednar**[2] & **Christian Huemer**[1]

[1]TU Wien, Institute of Information Systems Engineering, Favoritenstr. 9-11, A-1040 Vienna, Austria

[2]TU Wien, Institute of Material Technology, Building Physics and Building Ecology, Karlsplatz 13, Vienna, 1040, Austria

Abstract

Making sure that Building Information Modelling (BIM) models comply with building codes and contractual requirements is typically a time-consuming and overhead-heavy task. For this reason, there is a push for its automation. However, there are significant challenges to overcome. For example, most technical guidelines and contractual requirements are formulated in a natural language and would need to be translated into a *formal representation* for digital processing. Furthermore, there is no formal mechanism that enables the declaration, enforcement, and verification of compliance to those in a BIM model. In this work, we present a fundamental approach that makes the last three possible at any level of granularity, i.e., throughout the building's entire life cycle.

Introduction

Digital data models have become the backbone of information exchange in the Architecture, Engineering, and Construction (AEC) industry's domains. They have the potential to hold all data a single domain expert might need to perform a task, while at the same time, they have to be standardized enough to enable interoperability among multiple domains. This dichotomy drives the development of BIM models in two rather contradictory directions, towards greater flexibility *and* towards greater standardization, i.e., unification. This presents an implementation problem for any BIM software relying on the type-object paradigm and on type safety as a guarantee of correctness.

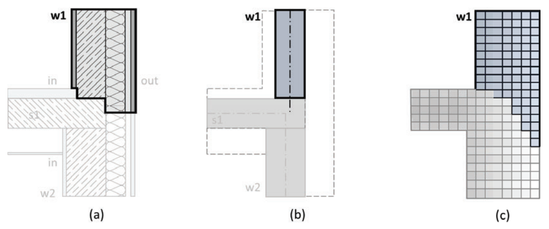

Figure 1: Different viewpoints of the concept "wall" in: (a) architecture, (b) structural engineering, (c) building physics.

First, this paradigm has no mechanism for managing abstraction, i.e., progressing from a more abstract to a more specific model. Second, multiple types corresponding to the requirements of multiple domains on the same model element (see Figure 1) can be implemented, e.g., via interfaces; however, those are defined by software engineers, typically indirectly following the specifications, generally in a natural language, of the domain experts. The domain experts themselves cannot define new types as they need them. What's more, software development cycles can range from one to multiple years, proprietary applications cannot be customised to all experts in the field worldwide, and experts themselves often have diverging views on their own domain and require different data models. This leads to the necessity for domain experts to be able to define new types for their models on demand, while at the same time, providing some sort of type safety to ensure correctness and reliable interoperability.

Figure 1 shows how much the view of different experts on the same concept can diverge, even concerning as "obvious" a concept as a wall. For example, architecture sees the wall's multiple functions and general extents, but is not overly concerned with its interface to its neighbours. Structural engineering, on the other hand, considers only the wall's load-bearing properties, all its joints to all its neighbours, and only the size of its load-bearing layer. Finally, building physics regards the wall as a part of an interconnected structure that conducts sound, heat, and water, both as a vapour and a liquid. It is understandable that such vastly different viewpoints require different data models. Taking the various calculation and simulation methods in use in each of these domains into account, the requirements on these models become even more complex.

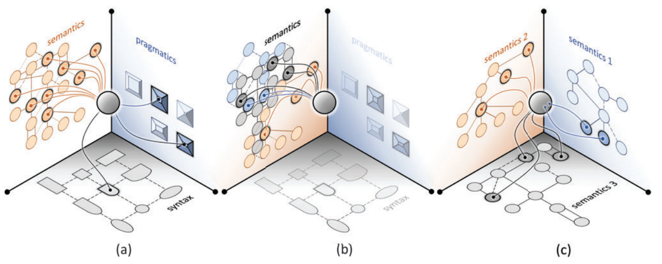

Figure 2: The dimensions of a digital domain model: (a) syntax, semantics, and pragmatics regarded separately, (b)-(c) separating independent semantic dimensions for better understanding.

Therefore, domain experts need to be able to not only define new types on demand, but also to keep the resulting data models interoperable *and* maintainable, i.e., to keep up with the rise in complexity. In this work we present a method and a metamodel capable of providing these features to any domain expert *without the involvement of a software engineer.*

Approach

A domain model generally has at least two dimensions, the syntactic, and the semantic [3]. In addition, there might be a representational dimension, a pragmatic dimension, and others. The semantic dimension itself might be heterogeneous enough for us to be able to split it into two ore more independent, i.e., orthogonal, dimensions, e.g., semantics of geometry and semantics of materials (see Figure 2). Regarding these dimensions separately can help remove some misunderstandings, e.g., when syntax and semantics get conflated.

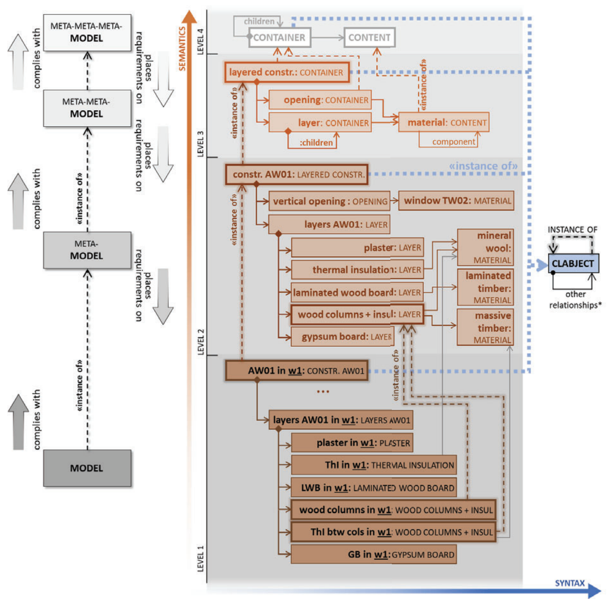

Figure 3: An example of four levels of semantic abstraction and two levels of syntactic support.

For example, in Figure 3 we can see the model of a wall in the 2d space of semantics (vertically in orange) and syntax (horizontally in blue). In our approach we utilize Model Driven Engineering (MDE), the Metamodelling Stack [2], and our own metamodel, SIMULTAN[1] [4], which implements the *clabject*, a syntactic construct capable of depicting both a class and an object, and the *instance of* relationship between them [1]. This works not just for two, but for arbitrarily many levels, in essence allowing a class to be an object relative to another class, hence the name *clabject*. In addition to the *instance of* relationship, the clabject enables all standard relationships between types, e.g., inheritance, association, containment, etc. All of this allows us to implement an arbitrary number of abstraction levels along the semantic axis. Each level utilizes the *instance of* relationship to place requirements on the level below as well as to comply with the requirements of the level above.

This gives us a stack of models [2], each fulfilling three roles for the models below it: it *declares* the valid types (or classes) that can be instantiated, it *enforces* compliance with those types, and it offers a way towards automatic *verification* of compliance with them. For example, the layered construction defined on semantic LEVEL 3 in Figure 3 requires all model elements that instantiate it to have layers, which can be observed in the instance **constr. AW01** one level below, on LEVEL 2. In the same way **AW01 in w1** on LEVEL 1 complies with the requirement of **constr. AW01** that there can be only the five layers defined in **constr. AW01**, even if, for example, **AW01 in w1** splits the inhomogeneous layer into its parts, columns and insulation, in order to model their geometry and behaviour separately.

References

[1] Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, Jul 2008.

[2] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. chapter Metamodeling, page 14. Morgan & Claypool, USA, 2 edition, 2017. isbn: 978-1627057080.

[3] Thomas Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):pp. 369–385, Dec 2006.

[4] Galina Paskaleva, Alexandra Mazak-Huemer, Manuel Wimmer, and Thomas Bednar. Leveraging integration facades for model-based tool interoperability. *Automation in Construction*, 128:103689, 2021.

[1]Actively developed by the Institute of Material Technology, Building Physics and Building Ecology at TU Wien.

Metamodels: Built-In BIM Compliance Checking (2)

Galina Paskaleva[1,2], **Thomas Bednar**[2] & **Christian Huemer**[1]

[1]TU Wien, Institute of Information Systems Engineering, Favoritenstr. 9-11, A-1040 Vienna, Austria

[2]TU Wien, Institute of Material Technology, Building Physics and Building Ecology, Karlsplatz 13, Vienna, 1040, Austria

Results

Here, we present one possible application of this approach on the use case of the AEC industry's concept of "wall" (Figure 3). Here, on LEVEL 4, we start with two very generic types, *CONTAINER* and *CONTENT*. On LEVEL 3, type **layered constr.** instantiates *CONTAINER*, which means that if the recursive relationship *children* is mandatory, then **layered constr.** *has* to contain at least one other instance of *CONTAINER* (see **opening**: *CONTAINER* and **layer**: *CONTAINER* on LEVEL 3). On this level, the layered construction definition is enriched by additional types that declare that its layers refer to materials (see **material**: *CONTENT*).

This allows us to instantiate one specific layered construction in the next level, LEVEL 2, **constr. AW01**. By virtue of being in an *instance of* relationship with type *LAYERED CONSTR.*, it is guaranteed to have layers. This particular wall has five: plaster, thermal insulation, a load-bearing laminated timber board, timber columns, and a gypsum fibre board. A further instantiation of this specific layered wall is shown on LEVEL 1, as **AW01 in w1**: *CONSTR. AW01*, i.e., the application of this wall construction to one specific wall, **w1**. This model design guarantees that **AW01 in w1** complies with all requirements placed on it by *CONSTR. AW01*, i.e., the material layers have the exact same sequence, thickness and material properties. However, in addition to those, the instance on LEVEL 1 can add a more detailed geometry, e.g., individual volume representations for each column in the inhomogeneous layer.

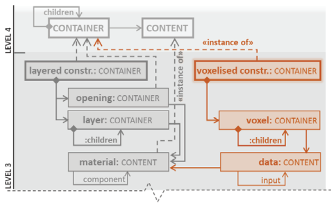

Figure 4: Adding semantic types to an already existing model: at high abstraction levels.

These dependencies become particularly useful, if we need to add model elements at a later stage, as shown in Figure 4 and Figure 5. For example, while a **layered constr.** is useful for *calculating* thermal transmittance, a **voxelised constr.** may be necessary for *simulating* the thermal transmittance. Figure 4 demonstrates not only that both types of construction can be instantiated from *CONTAINER*, i.e., be forced to comply with its requirements, but also that we can establish a correspondence between material and voxel data that will be a requirement of all models on the lower levels of the model stack. Similarly, we can add a new **AW01 in w2** on LEVEL 1 (see Figure 5), which, just as **AW01 in w1**, is an instance of *CONSTR. AW01*, i.e., the application of the same construction to a different wall, **w2**, that can put emphasis on other layers, if desired. For example, by instantiating all sub-layers of the thermal insulation from the same type on LEVEL 2, layer **thermal insulation**, we make sure that they are forced to fulfil the same requirements, i.e. of material and context, which removes multiple potential error sources from the transition from calculation to simulation of thermal transmittance.

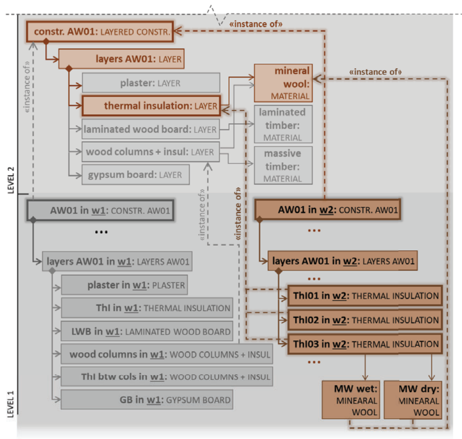
constr. AW01: LAYERED CONSTR.
«instance of»
layers AW01: LAYER
plaster: LAYER
thermal insulation: LAYER
laminated wood board: LAYER
wood columns + insul: LAYER
gypsum board: LAYER
mineral wool: MATERIAL
laminated timber: MATERIAL
massive timber: MATERIAL
LEVEL 2
AW01 in w1: CONSTR. AW01
AW01 in w2: CONSTR. AW01
layers AW01 in w1: LAYERS AW01
plaster in w1: PLASTER
ThI in w1: THERMAL INSULATION
LWB in w1: LAMINATED WOOD BOARD
wood columns in w1: WOOD COLUMNS + INSUL
ThI btw cols in w1: WOOD COLUMNS + INSUL
GB in w1: GYPSUM BOARD
layers AW01 in w2: LAYERS AW01
ThI01 in w2: THERMAL INSULATION
ThI02 in w2: THERMAL INSULATION
ThI03 in w2: THERMAL INSULATION
MW wet: MINERAL WOOL
MW dry: MINERAL WOOL
LEVEL 1

Figure 5: Adding semantic types to an already existing model: at lower abstraction levels.

Another feature of our approach, the separation of (nearly) independent semantic aspects into separate dimensions in a Common Reference Frame (CRF) is shown in Figure 6. As mentioned above, a construction can be modelled as consisting of layers or of voxels (see Figure 4). In Figure 6, we have the four semantic levels of the voxelised construction in blue under *semantic aspect simulation*, orthogonal to the semantic levels of the layered construction in orange under *semantic aspect construction*. A third aspect, the *geometry*, is another independent dimension, as shown in grey at the bottom. This enables us to maintain *model continuity* by, e.g., utilizing semantic LEVEL 3 to model a wall in the design phase as **wD1**, and semantic LEVEL 4 to model its more detailed version in the tender document phase, as **wT1**. This guarantees that all requirements placed on **wD1** are enforced in **wT1** as well. At the same time, despite its complexity, the domain model remains both consistent and maintainable due to the separation of semantic aspects and abstraction levels.

We have tested this approach on real world examples encompassing not just architecture, building physics, and structural engineering, but also building services and automation. The separation of semantic aspects enabled us, for example, to handle the topology of a ventilation network separately from its other properties, e.g., its connection to the data network supporting automation [4].

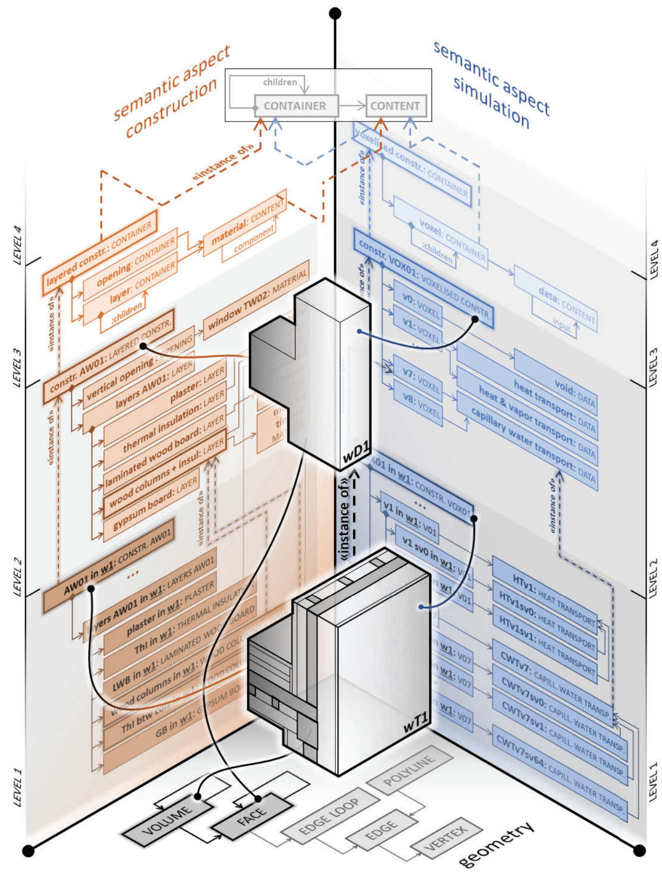

Figure 6: Multiple semantic dimensions and multiple abstraction levels: enabled by metamodels.

Contribution

- The metamodel SIMULTAN implements a syntactic tool, the clabject, which allows us to produce an arbitrary number of semantic levels of abstraction in any BIM model.
- This meta-model stack allows each model to declare, enforce and formulate rules for the verification of domain-specific requirements for all models on the lower abstraction levels.
- Dependencies between different domains can be declared on high abstraction levels and, subsequently, enforced on all lower abstraction levels, thereby making collaboration between domains in the AEC industry less error-prone.
- Different semantic aspects can be modelled separately, which reduces the complexity and increases the maintainability of the large BIM models typical for the AEC industry.

References

[1] Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, Jul 2008.

[2] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. chapter Metamodeling, page 14. Morgan & Claypool, USA, 2 edition, 2017. isbn: 978-1627057080.

[3] Thomas Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):pp. 369–385, Dec 2006.

[4] Galina Paskaleva, Thomas Lewis, Sabine Wolny, and Thomas Bednar. Simultan as a big-open-real-bim data model - evolution of virtual building from design through construction into operation phase. In *21st CIB World Building Congress. Constructing Smart Cities*, pages 1116–1126, Jun 2019.

[5] Galina Paskaleva, Alexandra Mazak-Huemer, Manuel Wimmer, and Thomas Bednar. Leveraging integration facades for model-based tool interoperability. *Automation in Construction*, 128:103689, 2021.