https://doi.org/10.1016/j.aei.2022.101664

Graph-based version control for asynchronous BIM collaboration

Sebastian Esser[a,*], Simon Vilgertshofer[a], André Borrmann[a]

[a] *Chair of Computational Modeling and Simulation, School of Engineering and Design, Technical University of Munich*

Abstract

Collaboration and communication are two essential aspects of Building Information Modeling (BIM). Current practice and international standards implement BIM collaboration on the basis of domain model federation where loosely coupled models are managed as separated files and coordinated in a mostly manual fashion. The concept has severe limitations regarding concurrency and version control, as the granularity of change tracking remains on the level of complete files and does not reach individual model objects. Due to this lack of change traceability, high manual effort for the subsequent coordination across the domains is generated. These limitations can be overcome by implementing modern approaches of digital collaboration based on object-level synchronization, widely denoted as BIM level 3. This paper presents a sound methodological basis for object-based version control by (1) representing the object networks of BIM models as formal property graph structures and (2) describing changes of the model by graph transformations. Consequently, modifications can be transmitted as graph transformation rules which are subsequently integrated on the receiving side, thus achieving object-level synchronization. The paper provides the underlying theory of describing model changes by means of graph transformations and demonstrates its benefits using the example of domain models implementing the Industry Foundation Classes (IFC) as their underlying data model.

Keywords: Asynchronous collaboration, object-based version control, BIM Level 3, Common Data Environment

*sebastian.esser[at]tum.de

1. Introduction

The design, construction, and maintenance of buildings and civil infrastructure are among the most essential societal tasks. Particularly in large projects, such as the design of major transportation infrastructure or highly equipped buildings, a very large number of experts and domains is involved to generate cost-effective yet efficient solutions for challenging environments. To accomplish solutions for complex multi-disciplinary engineering problems with contradicting objectives such as cost reduction, environmental preservation, and responsible resource consumption, planning and design tasks often require several iterations to meet all requirements from various domains and parties. Through these iterations, the design of the built asset to be constructed is successively refined and elaborated. Iterative processes entail that information which is shared at one point in time might no longer be valid in the future. Hence, keeping an overview of an evolving information set becomes a cumbersome task. Therefore, powerful management tools are essential to coordinate all complex dependencies and relations among all involved parties.

To reliably represent and exchange design information, the methodology of Building Information Modeling (BIM) has been increasingly adopted by the Architecture, Engineering, and Construction (AEC) industry throughout the last years [1, 2, 3]. As the advantages of BIM over conventional drawing-based practices are overwhelming, a rising number of public authorities and contractors demand BIM-based collaboration. To streamline BIM information management, a number of standards have been defined. Among the most important is ISO 19650 [4] that specifies the principle of model federation and describes the *Common Data Environment* (CDE) as the technological solution for data management in BIM projects [5, 6, 7]. At its heart, the principle of model federation relies on discipline-specific partial models (such as Architecture, Structural, or HVAC models) that are created and maintained independently, but coordinated in well-defined intervals to achieve a coherent overall design solution. The main motivation for federated model approaches lies in the fact that legal responsibility for disciplinary design decisions must remain with the respective author. In addition, the practice has shown that asynchronous collaboration is much more appropriate for cross-enterprise BIM design projects than synchronous collaboration, as it allows for the desired independence of the involved partners in terms of time and resource planning.

According to ISO19650, the domains models and associated documents are managed and exchanged by means of so-called *Information Containers*. Implementing this approach, current best practise relies on transmitting entire and complete domain models stored in files each time a model version is shared. In consequence, the changes applied to the respective domain model are neither explicitly propagated nor tracked or managed by the CDE. Therefore, it remains the responsibility of the stakeholders to manually find the applied changes and check for potential inconsistencies with their own domain models. This severely limits the available technical support for model-based collaboration and increases the risk for inconsistencies.

1.1. Problem statement

Even though a file-based data exchange suits the traditional fashion to store information on a hard drive, there are significant limitations when it comes to fulfilling the requirements of high-frequency model-based collaboration taking into account the large size of model data and the various dependencies across the design disciplines. Combining the situation of multiple versions being created and deployed as files throughout design phases with the large number of tasks that must be considered, the manual tracking of changes can be overwhelming and results in repetitive, error-prone processes. A much more effective approach can be achieved by exchanging only the differences between the two model versions. Incremental updates enable precise tracking of changes, allowing engineers to concentrate on these updates and the required modifications of their own discipline models. In addition, it opens ways for automated (yet controlled) updates across disciplines wherever a formal description of dependencies is possible, without violating the principle of discipline-oriented ownership and responsibility. As a side effect, data traffic produced to deploy a new version is drastically reduced as the ratio between modified components and the overall number of elements in a model is typically quite low.

While there are established version control systems (VCS) such as Git or Subversion, which implement the concept of incremental changes, they are dominantly based on tracking and updating textual representations like program code. Knowledge representations used in the AEC industry, however, are complex, highly connected object networks that are much better represented by graph structures than by serialized textual representations. Approaches that formally describe engineering knowledge by means of graphs have been already investigated by a vast number of publications and provide a promising basis to implement object-based update tracking for BIM-based workflows.

1.2. Contribution

The work presented in this paper aims to overcome the limitations of file-based collaboration by developing graph-based methods for update tracking and federation in BIM models. A novel method for version control of BIM models is proposed that operates on object level and extends existing versioning concepts by introducing graph-based mechanisms. In particular, object models are proposed to be represented by directed, attributed graphs and modifications are formally described by graph transformations.

In terms of the system architecture, we assume a centralized hub for data management, which various clients can connect to and use to share their domain-specific models. For synchronizing the local model version with the central one, however, mere update patches are transferred instead of complete model files. These update patches contain the corresponding graph transformations. The concept of federated models, asynchronous collaboration, and eventual synchronization of all domain models into a centralized coordination model is maintained and implemented in accordance with existing standards and guidelines.

While the concepts for graph-based model versioning are generic, it is not the aim of the paper to define a new all-embracing data model that fits multiple domains. Rather, the developed method features existing and well-developed data models targeting highly specialized branches of the AEC industry and map their data sets onto a common graph structure.

In summary, the main contributions to be presented in this paper are:

- The concept of graph-based version control applied to BIM collaboration.
- A generic representation of BIM instance models by means of attributed, directed graphs.
- A graph-based algorithm that computes the difference between two versions of a BIM model on the basis of their graph representations.
- A mechanism to exchange and integrate model updates by means of formal graph transformations.

The paper is organized as follows: Section 2 discusses related work in the field, section 3 describes the developed method from a theoretic point of view. Section 4 presents a case study to provide a proof-of-concept, section 5 contains the discussion and section 6 closes the paper with a summary of the main findings.

2. Background & related work

The ongoing development of digital technologies has equipped various industry sectors with new opportunities to enhance data exchange and collaboration. In this regard, especially approaches to collaborate across domain borders expose a great potential. This section provides an overview of existing versioning systems suitable for textual knowledge representations. Furthermore, existing approaches for representing complex object networks by means of graph theory and graph transformations are introduced and put into the domain context of the AEC industry.

2.1. Representations of engineering knowledge

Domain experts involved in the design and engineering process of building and civil infrastructure assets capture knowledge typically in the form of discipline-specific data representations that are referred to as BIM models [8]. Such instance models contain semantic and geometric information of the designed asset from the viewpoint of the respective domain and are composed of a set of model elements reflecting a physical, logical, or contextual item and their mutual relationships. A BIM model instantiates a data model, which defines available types, classes, and their relationships. Various data models exist to ensure a common understanding of the knowledge shared within an instance model. These specifications implement the `Meta Object Facility (MOF)` definitions [9, 10]. The MOF specifications standardized by the Object Management Group (OMG) distinguish between instance data (M0), data model (M1), metamodel (M2), and metametamodel (M3). In the context of the paper at hand, the terms *instance model* and *BIM model* refer to the concepts stated in M0. The underlying structure, which abstracts the given real-world problem from a domain-specific perspective, is defined as *data model*. The abstraction of a data model in its generic items such as attribute types and relationships is defined in a meta model.

The exchange domain models is currently realized by uploading entire files to so-called Common Data Environments (CDEs) [11, 12, 4]. Yet, individual objects inside a instance model are typically not accessible for the CDE, not analyzed further, and hence not subject to versioning and concurrency control. In consequence, the granularity of versioning remains on a rather coarse level of entire domain models. In order to facilitate an object-based versioning of the populated instances, a general understanding of the structures and common representation types is necessary.

In most cases, a textual representation of the instance model is used to translate each object from the internal memory of the authoring software into a textual representation, which is then transferred to the project's CDE. Open, vendor-neutral instance models typically follow common encoding mechanisms (such as XML, JSON, STEP Physical File), whereas proprietary file formats typically implement binary representations. While proprietary file formats are still in wide use for various reasons, in the scope of this paper, specific emphasis is put on vendor-neutral information exchange. At a first glance, approaches for versioning textual representations appear promising to identify, track and manage model modifications.

2.2. Existing version control systems

The versioning of structured information is a relevant issue in many industry branches. Specifically, in the field of software development, various methods, protocols, and systems exist that enable distributed version control of text files [13]. Prominent examples of VCS are Apache Subversion[1], Mercurial[2] and Git[3] among others. In most approaches, a central database stores the global history of change events, integrates incoming modifications, and allows a user to clone the entire history with all incremental changes to a local machine. If changes are ready to be shared with others, the user commits the local state and pushes it to the central database again. The chain of update messages forms the entire history of the project. Incoming updates can be integrated automatically if they do not create any conflicts with existing or concurrent local changes. Only in case of conflicts, the user needs to resolve them and choose the desired content manually [14].

Even though such version control systems are well established, they show significant limitations when it comes to the management of complex object networks given in BIM models. Most of the time, serializing connected information into textual representations requires the introduction of additional identifiers to specify dependencies between objects. These identifiers are not part of the actual domain to be described in the model and are simply due to the need to represent associations between instances in a textual manner. Engineering knowledge can be serialized into instance models in a completely different order depending on the serialization strategy, resulting

[1]https://subversion.apache.org/
[2]https://www.mercurial-scm.org/
[3]https://git-scm.com/

in seemingly diverging files which in fact represent the same content. Thus, pure text-based comparisons are not capable of reflecting the complex and highly interconnected object networks.

In the AEC industry, a number of researchers have already investigated the representation and exchange of modifications on specific data sets. For example, Koch and Firmenich [15] have presented a comprehensive versioning approach for procedural building representations. Their approach is based on representing both, design states and state transitions, thus enabling the exchange of change-oriented information by means of design steps denoted as modeling operations. Even though the general vision is highly related to the approach, the paper at hand focuses on expressing modifications by graph transformations applied on the model. On the contrary, the work of Koch and Firmenich focuses on a procedural description of the model creation process and the subsequent versioning of them. A similar approach is taken by Vilgertshofer and Borrmann [16], which however focuses on the generation of models. Both publications suggest a general validity of the approaches used, but further consideration of the framework conditions is required to apply them in the context of BIM models featuring domain-specific requirements.

Apart from research initiatives, multiple software vendors have identified cloud computing and collaboration systems as crucial to their product portfolio. Prominent vendors like Autodesk, Trimble, Graphisoft, and others provide cloud-based solutions to allow working on data sets in a collaborative manner [17, 18, 19]. Their approaches, however, feature only vendor-specific file formats and data models. A good example is Graphisoft's BIM cloud service, which was formally know as Graphisoft Delta Server, implementing the idea of transmitting deltas (model differences) for synchronizing a distributed system [20]. To the best knowledge of the authors, these products do not provide a generic approach that can be applied to a multitude of data models used for interdisciplinary data exchange. Besides, the technological foundations used in the collaboration systems are often not publicly documented, which makes it complex to argue about their applicability to data structures other than their internal ones.

By contrast, the Speckle platform presented by Poinet et al. [21] aims to provide an object-based collaboration system for BIM data by implementing direct communication between senders and receivers. The overall data exchange relies on a generic object definition called *SpeckleObject*, which carries dynamically created attributes. Therefore, the approach correlates with fundamental concepts of dynamically typed languages such as Python and enables the user to create and modify the attributes of a certain object in a flexible manner. Even though Poinet et al. [21] claim to support subsets of vendor-neutral data models in their system, a common understanding of Speckle's object architecture is required at all sending and receiving systems. Subsequently, the approach appears promising but also contains the risk to create yet another data exchange standard in the industry connecting only a selected set of systems.

To allow object-level versioning of complex object networks in a generic manner (i.e., independent of a specific data model or vendor), novel methods are necessary to detect modifications between two versions directly in the object network instead of computations on their text-based serializations. Therefore, the next paragraphs reports on existing approaches to reflect BIM models in graph structures.

2.3. Graph representations of object networks

Capturing and representing engineering information by means of graph structures has been of research interst in many industries and domains. A major advantage of graphs is the ability to model dependencies and relationships between individual objects in a generic manner [22]. Physical and logical assets can be abstracted in a flexible manner without having to consider application-specific conditions for each object or relationship. Therefore, graph structures are often used to enrich information resources by adding extended knowledge to them. Application scenarios can range from optimisation problems (e.g., agent routing through complex networks [23]) over computational synthesis tasks [24] to data fusing applications that aim to detect implicit dependencies in large data sets [25, 26, 27].

All graphs $G=(N, E)$ consist of a set of nodes N and edges E, with each edge connecting a number of nodes (usually 2) in a directed or undirected manner. Depending on the specific domain, nodes and edges may carry additional information in the form of attributes and labels supporting efficient querying, sorting, and other computational features. Subsets of graphs are called sub-graphs or graphlets whereas the term *pattern* is used to specify a graph or graphlet (e.g., to be used in a query statement).

Graph representations for structured data sets can be divided in two major categories. Approaches related to Linked Data principles represent the object networks using a *triple centered* approach where each triple is composed of a subject, a predicate, and an object [28]. A triple can model an attribute belonging to an object (e.g., "the material of a wall is concrete") or describes a relationship between two objects (e.g., "the wall is contained in a building"). In terms of the corresponding graph, each triple is represented by an edge. Each information item (objects but also attributes) is represented by a node. Correspondingly, graph structures representing complex models consist of a large number of nodes and edges. In contrast, entity-centered approaches are more closely aligned with the principles of object-oriented analysis and design (OOAD), where each object has a set of attributes, which describes the current state of the instance. In the corresponding graph structure, each object carries its attributes as weights and edges are only used to model relationships with other objects.

A number of researchers have investigated the representation of building information models as graphs. The approaches typically rely on (1) representing first-class object types such as walls, windows, doors as nodes and (2) representing their topological or functional relationships with the help of edges between the corresponding nodes. The presented approaches vary in the choice of first-class objects and considered relationships, as they follow different purposes or views on buildings [29, 30, 31, 32].

Various approaches exist to define, store, exchange, and deploy triple-based data. One prominent specification is the *Resource Description Framework* (RDF), which defines a common grammar or schema definition on how triples are formulated. Like XML, the RDF standard is maintained by W3C [4]. The application of RDF knowledge representations in the AEC sector has been addressed by numerous publications [33, 34]. Additionally, many researchers have investigated query mechanisms of BIM models modeled in RDF techniques [35, 36, 37]. Oraskari and Törmä [38] have applied diff computation mechanisms on RDF graphs that were produced from IFC instance models. Further, Rasmussen et al. [39] propose the definition of an ontology to describe the evolution of RDF graphs in a formal and computer-readable manner.

As RDF data sets often evolve dynamically, Roussakis et al. [40] have presented a method to identify and analyze by calculating the delta between two given RDF graphs. As the definition of an alteration is highly dependent on the specific knowledge reflected within an RDF graph, configurable *SPARQL* queries are used to identify changes among two versions. Furthermore, Singh et al. [28] use the expression *dataset dynamics* to describe constant changes

[4]https://www.w3.org/RDF/

applied to an information resource. Their DELTA-LD change model combines resource and triple-centered views into a common change management model for linked data sets. Bobed et al. [41] have investigated version control systems of RDF-based knowledge representations and have defined two major states: An *update* captures a mutation applied to a small subset of the RDF graph whereas a *snapshot* freezes and stores the current state of the knowledge system.

As already addressed, a major disadvantage of storing BIM models in triples is the large file size compared to entity-centered approaches. To overcome these limitations, Zhao et al. [42] have presented an approach to merge data sets facilitating the IFC data model based on graph structures. Even though their approach appears promising for coordination tasks and decentralized collaboration, their approach is dependent on the IFC data model and cannot be transferred to other data representations used in the AEC industry. Same applies for the research by Shi et al. [43], who have proposed an approach that allows detecting differences between two BIM models based on a similarity metric. Their system runs a normalization on all instances stored in the model first and calculates a similarity score afterward using a recursive depth-first search. Unfortunately, the resulting similarity rate is presented as a mere scalar value, which does not provide access to the altered objects within the model. Looking on graph similarity from a generic perspective, the concept of Maximum Common Subgraph (MCS) by Schultheiß et al. [44] supports the identification of repetitive graph structures that appear in multiple graph structures. Further research tackling aspects of MCS was conducted by Wang and Maple [45], Bunke et al. [46], Conte et al. [47].

2.4. Graph transformations

In addition to graph querying and filtering, graph transformations are a very relevant field of research for the scope of this paper. The concept of graph transformation (also denoted as graph rewriting) has its roots in formal graph theory and its sound mathematical definitions. The general idea of graph rewriting is the mutation of a given host graph H into a resulting graph H' by applying a set of transformation rules r. The formulation of rules builds upon mapping functions that express relationships between nodes and edges of different (sub-)graphs [48]. The general principle of a graph transformation is depicted in figure 1.

Each transformation rule consists of a pattern graph L and a rewrite graph R. The pattern graph specifies the structure, which the rule should be applied

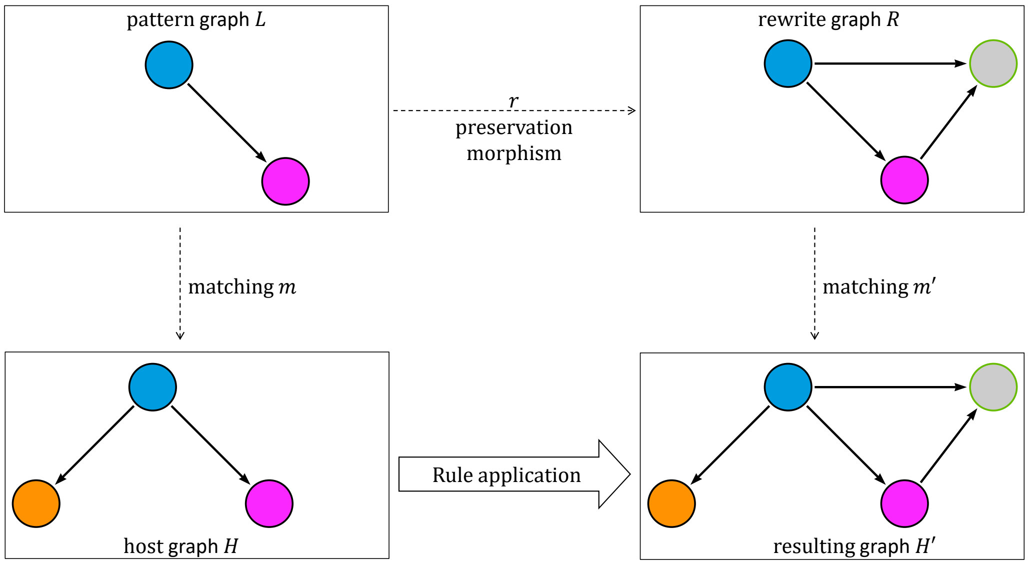

Figure 1: An SPO-based graph transformation rule inserting a new node connected by two edges. Each color indicates a unique node, which makes the specified patterns occur exactly once in the host and resulting graphs H and H'. (inspired by Blomer et al. [49] and Vilgertshofer and Borrmann [16])

to. Once the pattern graph is identified in the host graph (i.e., a subgraph exists in H that is homomorphic to the pattern graph L), the rule is applied by replacing the pattern L by the rewrite graph R. Well-established transformation techniques are the Double-Push-Out (DPO) approach among other techniques such as Single Push Out, Sesqi Push Out, and others [50, 51, 52]. Even though all methods have in common the goal of transforming a given host graph into a modified state, the Double-Push-Out approach consists of two separate pushout operations. This characteristic requires a more precise definition of how inclined edges to a inserted or removed node should be handled during the transformation. For detailed explanations and algebraic background information, the reader is advised to read the publications of Buchwald [53] and Blomer et al. [49].

Starting with the pre-condition of a DPO transformation rule, the *Left Hand Side* (LHS) L specifies a pattern that the transformation system is searching for in a given host graph H. If an occurrence of the pattern L is detected in H (i.e., L is a homomorphic subgraph to H), all nodes and edges contained in L but not in the interface I are removed from H, which results in the context graph D. The interface I is defined such that I is isomorphic to L, which can be seen as an intermediate state after applying the remove operations. Accordingly, the *Right Hand Side* (RHS) specifies all nodes and edges that are inserted to achieve the intended resulting graph denoted as H'. The interface graph I is homomorphic to a subgraph of R describing the Right Hand Side of the rule. All nodes and edges present in R but not in I will be inserted to achieve the resulting graph H'. Accordingly, the graph R is naturally homomorphic to the rewriting result H'. Figure 2 depicts the involved graphs and patterns including all morphism statements necessary for a DPO transformation rule. In this example, the first transformation l removes the edge labeled with 2 connecting the pink and the green node. The removal is achieved by the preservation morphism l between I and L. Accordingly, the edge labeled with 3 is inserted to connect the green and the blue node using the preservation morphism r between I and R.

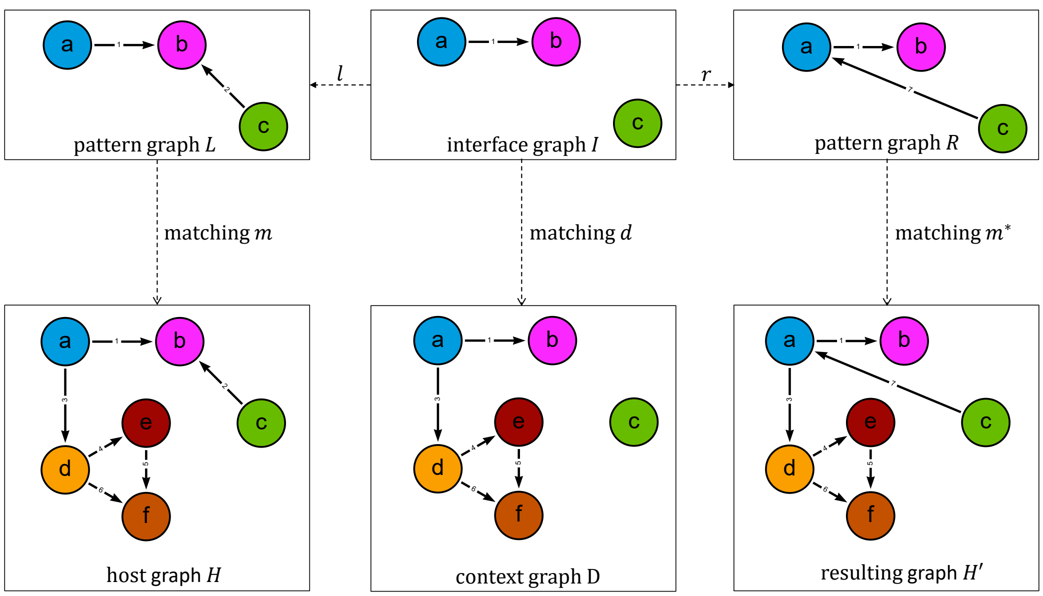

Figure 2: The DPO approach describing a graph transformation by a Left Hand Side L, an Interface pattern I, and the Right Hand Side pattern R. Each color indicates a unique node, which makes the specified patterns occur exactly once in the host and resulting graphs H and H'. (Figure inspired by Kniemeyer [54] and Javed [55])

Besides use cases like refactoring and optimization problems, various research activities have involved the application of graph rewriting in the scope of parametric geometry [16] or building models [56]. Helms and Shea [24] and

Königseder [57] have investigated the field of computational design synthesis based on formal graph grammars. Even though their research grounds on related concepts of set and graph theory, their approaches aim to generate solution spaces containing a large set of options an engineer could consider for his design task. The transfer of a modification applied to a BIM model, however, should result in exactly one solution, namely the updated state the model author has generated. Nevertheless, graph transformation is suitable tool to express exactly this change. The exact formulation of a transformation rule depends on the graph meta model, which provides a suitable vocabulary to express a specific modification. The meta model used to implement the proposed version control systems for BIM models will be described in section 3.2.

2.5. Identified research gap

In summary, the literature review proves an enormous interest in versioning control systems of structured data across various engineering disciplines. However, none of the existing techniques can be used for the incremental version control of BIM models based on their underlying object networks. A large number of research items deals with the reflection of information produced during a design process in graph structures. Yet, only a few approaches address issues related to evolving object structures in a domain-independent manner or use flexible graph meta models that are capable of reflecting various data models.

The paper at hand investigates the combination of established principles of common graph representations and how they can support incremental versioning systems for data models used in the AEC industry. The underlying hypothesis is that graph theory is well suited to capture the complex and highly interconnected object networks of a BIM model, and that graph transformation is a suitable mechanism to describe modifications of these structures. Contrary to many existing approaches, the applicability of the developed system for a wide range of data models is a key objective. The following section introduces a graph meta model, which reflects commonly used data specifications generically and discusses the representation of modifications by graph transformation rules. Furthermore, it will provide an illustrative example demonstrating the application of the methodology.

3. The developed graph-based diff-and-patch method

3.1. Overview

A novel approach that supports the formal identification of updates (*diff*) in BIM models and their integration into federated copies of a model (*patch*) is required to overcome the limitations of collaboration systems currently used in the AEC sector. To ensure the successful integration of the proposed approach in existing workflows, all concepts build upon the assumption of asynchronous collaboration using loosely coupled discipline-specific BIM models. Hence, each domain works in its specialized software environment and provides discrete versions of the developed domain model to a central project repository to share it with other collaborators. This involves a synchronization step where all deployed copies (e.g., to project platform or other designers) of the model are updated such that it mirrors the most recent state of the model stored at the author.

In the developed method, each time a new version of a BIM model is produced, the unchanged parts are identified that have already been made available in the previous version. Subsequently, only the modification (i.e., the transformation rule) necessary to update outdated replicas of the previous version is deployed to other project stakeholders. The proposed method is generic in the sense that it is agnostic to specific data models, i.e., applicable to a wide range of existing object-oriented data models. This is realized by employing graph structures to represent the complex object networks of BIM models. For representing the changes between two versions, the concepts of maximum common subgraphs (MCS) and graph transformations are applied.

Figure 3 depicts the envisioned data flow between a sender and a receiver. In most cases, the sending side is formed by a modeler's workstation and the receiving side is the central repository. However, de-centralized architectures are also supported. Once a model author has developed a new shareable state of the domain model, the *diff* between the latest deployed version and the new model revision is computed by comparing the graph representations of both versions. The comparison results in the definition of a graph transformation rule p, which expresses the applied modification. Contrary to classical graph transformation problems, the transformation rule (i.e., the pattern describing L, I, and R) is not known a priory and is computed based on the initial and the updated model graphs. On the receiver side, an interpreter applies the transformation rule to the locally stored graph to obtain the same updated state that the sender has created. The resulting graph can be translated back into a file-based representation to ensure compatibility with existing software applications, which provide import modules for the particular data specification.

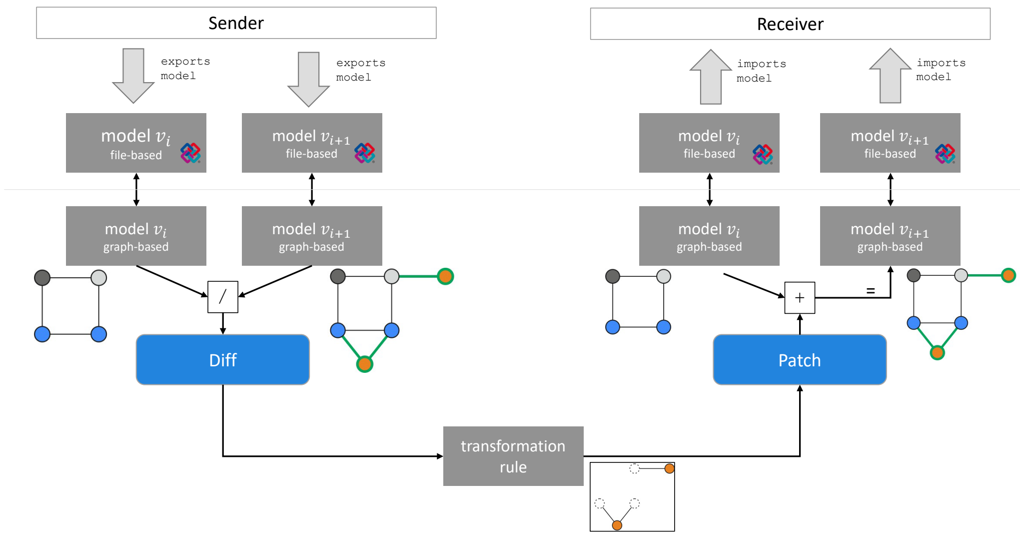

Figure 3: System architecture for object-based modification exchange between a sender and a receiver using graph transformation rules

3.2. Translation of object-oriented information into graph structures

As most information exchange scenarios employ well-defined data models that follow object-oriented paradigms, we adopt the approach of modeling object-oriented domain models as attributed, directed property graphs introduced by Hidders [58]. Like any other graph structure, property graphs consist of a set of nodes N and a set of edges E. The core concept of this approach lies in representing each class instance by a single node of the graph. Accordingly, associations between two class instances are modeled as an attributed edge in the graph. In a property graph, to both nodes and edges properties can be assigned. Each property is represented as a key-value pair, such as {weight: 10}, {color: 'red'}, {name: 'Alice'}. Depending on the chosen graph system, property graphs might require the definition of a graph meta model describing available labels and attributes of nodes and edges. As stated earlier, the method presented here aims at providing a generic, schema-independent approach. At the same time, a basic structure of the underlying data model is assumed that helps to identify changes across versions. Hence, the graph meta model employed to represent BIM models as graphs consists of three different node types and one edge type: `primary nodes`, `secondary nodes`, and `connection nodes`.

By definition, instances of `primary nodes` have a unique identifier attribute. This kind of uniquely identifiable instances exists in any known data models in the BIM context. By contrast, `secondary nodes` reflect all class instances that specify information without having a unique identifier. The third type of nodes is denoted as `connection nodes`. These nodes represent one-to-many or many-to-many relationships among instances. To identify all nodes belonging to a specific version of a domain model, each of these items gets an additional label attached indicating the timestamp of the creation date (typically defining, when the domain model is exported from the authoring tool). Therefore, the graph structure reflecting a particular version of a domain model can be easily identified. All attributes of a class instance are stored in the property set of the node representing this instance. Additionally, the *EntityType* attribute reflects the name of the class as a textual value. Associations between classes are realized by edges between two nodes. To define the association properly, each edge carries an *relType* attribute reflecting the association name. In case an association references a set of instances, an additional counter attribute is attached to each edge to preserve the order of the set.

In all of the investigated data schemata (including the Industry Foundation Classes (IFC), LandXML, CityGML, PlanPro, RailML and others), class instances with unique identifiers establish a semantic skeleton, to which several "resources" are bound that regularly do not possess unique identifiers. These resources are defined according to domain-specific requirements and may represent geometric shapes, material associations, costs etc.

Figure 4 depicts a fictitious example of a data model, an instantiation and the resulting property graph structure reflecting the instances. The data model is described with the modeling language EXPRESS. The schema definition in the upper left corner defines entities (i.e., classes without methods).

The mapping of the given domain model onto the proposed graph structure follows the rules previously explained:

1. Each node reflects one entity instance.

2. All instance attributes are directly attached to the respective node reflecting the specific instance whereas associations are modeled as directed graph edges.
3. Each edge carries the attribute name from the class, from where the association was initialized.
4. Both instances of the `ShapeElement` entity are represented as a primary node depicted in blue color as they possess a unique identifier attribute.
5. All remaining instances are reflected by secondary nodes illustrated in yellow.

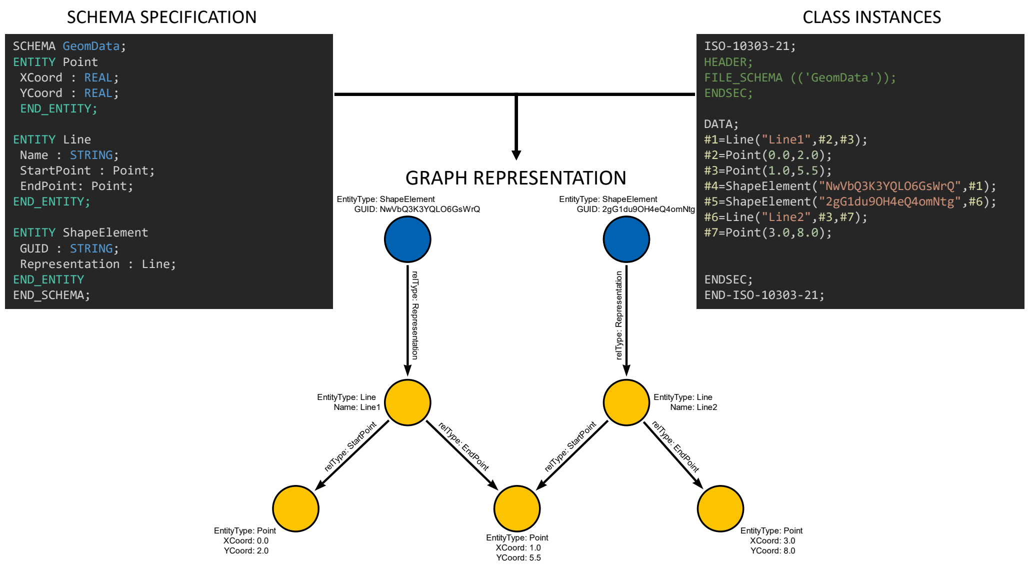

Figure 4: Correlation between a data schema in EXPRESS, instances serialized into a STEP Physical File (SPF) representation, and the resulting property graph

The next paragraphs describe how two versions of a model are compared and how the subsequent patch is created.

3.3. Diff Calculation

In principle, there are two options for identifying the changes between two versions of the BIM domain model to formulate the patch:

1. Tracking changes by listening at the BIM authoring application through API callbacks.

2. Comparing two versions of files exported from the BIM authoring application into an open exchange format.

For this study, we decided for option 2 as it is more generic and allows for integrating a large number of authoring tools without specific adaptions. As, however, option 1 is more compelling from a conceptual point of view, we will investigate it in future publications. Hence, the developed method covering option 2 is based on determining the transformation rule by comparing the initial and the updated graph representation of a given BIM model. Applying the concept of the DPO graph transformations depicted in figure 2 to the outlined situation, the host graph H reflects the BIM model in its initial state whereas the resulting graph H' stands for the updated version of the BIM model. Accordingly, the context graph D reflects all instances and associations that are present in the initial and the updated model version, thus, reflecting all unchanged parts of the BIM model. Hence, the Diff computation aims to define a transformation rule based on the host and the desired resulting graph, which is contrary to many classical graph transformation problems where the host graph and the transformation rules are known.

For the comparison between two versions of the graph, the concept of *Maximum Common Subgraphs (MCS)* assists in identifying all parts of the graph that remained unchanged [44, 45, 46, 47]. The MCS algorithm traverses through both graphs and applies a depth-first approach. For each pair of matched nodes, the algorithm finds correspondences in the sets of their direct children, and so on recursively. The recursion considers a node of the initial version as potentially equivalent to a node of the updated domain model if both nodes are associated with the previously matched node pair using the same *relType* attribute. Figure 5 illustrates a recursion step and the subsequent semantic comparison of two nodes.

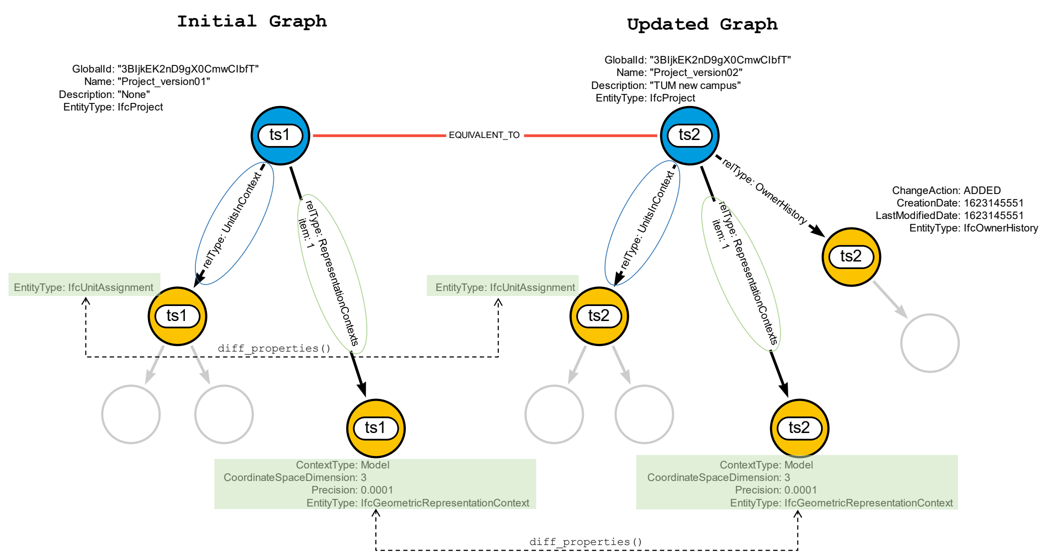

Figure 5: Recursion step starting from a pair of `PrimaryNodes`: Two nodes $n_{init}, n_{updated}$ are considered as equivalent if their inbound edges share the same edge attributes. The node labels $ts1$ and $ts2$ indicate the affiliations to a particular version of the BIM model. The identified equivalence between two nodes is reflected by an undirected edge labeled as *EQUIVALENT_TO*

To start the recursive traversal, the property of unique identifiers assigned to each primary node is utilized to find two nodes that represent the same object in the initial and the updated domain model. The algorithm stops the depth-first search if a leaf node is detected (no edges pointing outwards) or a node pair has been already marked as equivalent, respectively. Nodes that have been already considered as equivalent are connected by an additional undirected edge connecting them across the version-specific graphs. Doing so, it is straight-forward to identify all nodes from the graph system, which belong to the current version but have not been matched with a node of the previous version.

Hence, nodes reflecting instances in the initial graph but do not have a *EQUIVALENT_TO* edge indicate a remove operation whereas nodes in the updated graph representation lead to an insert operation being applied. These situations are denoted as *structural modifications* which require the formulation of full graph rewriting patterns. Apart from them, nodes might be detected that have diverging properties but have equivalent edges to all neighboring nodes. These changes are treated as *property modifications*, are comparatively simpler, but will also be represented as graph transformation rules.

On this basis, further processing of structural changes is necessary to identify the entire graphlet removed or inserted that is referenced by the node found by the diff algorithm . Figure 6 schematically illustrates the general situation of an insertion.

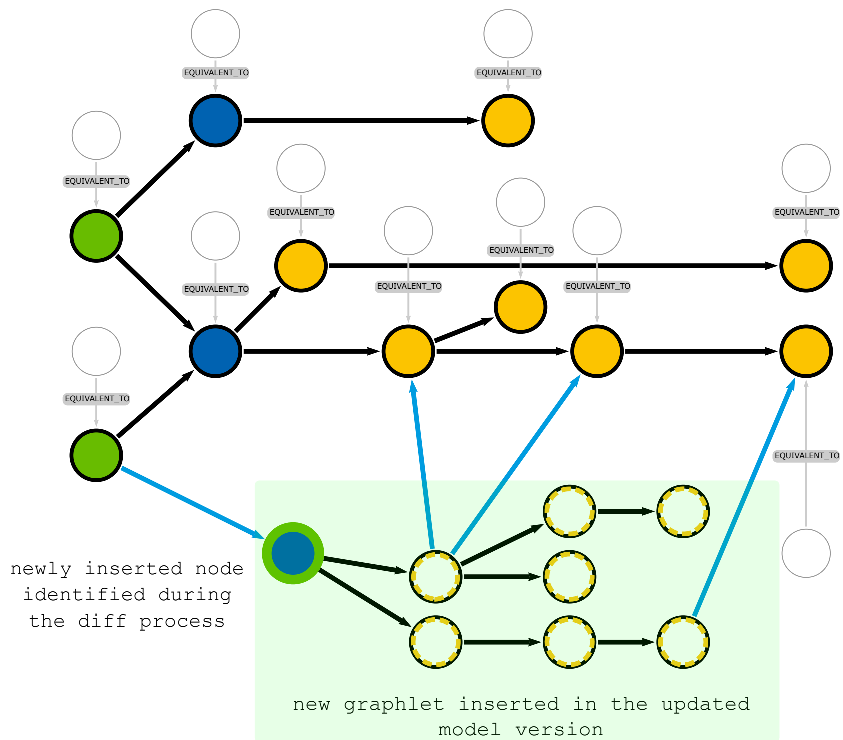

Figure 6: Schematic illustration of diff result and the patch formulation: Nodes with an equivalent counterpart in the compared model version have an *EQUIVALENT_TO* edge. The blue node with green margin has been identified as added, therefore, further processing is necessary to identify the graphlet inserted (depicted with dashed yellow margins) and the edges connecting the inserted graphlet to nodes that are present in both versions of the graph.

The node depicted with the green margin has been identified as inserted in the updated version of the graph. With this node being identified, the graphlet describing the inserted subgraph is identified from the graph. This graphlet contains all nodes and edges the resulting patch must insert into the host graph when applying the transformation rule. Furthermore, some of the newly inserted nodes have edges to nodes that have been previously existing, i.e., have an *EQUIVALENT_TO* relationship indicating that the node exists in both versions of the graph.

3.4. Representing patches as graph transformations

For structural modifications, it is essential that the transformation rule does not only describe the actual graphlet to be inserted or removed, but also its context or embedding into the object network that has been available already in the previous version. In the DPO concept, the insertion or removal of graphlets is formally described through the preservation morphisms l between I and L, and r between I and R, respectively. As explained in Section 2.4, nodes and edges are kept unchanged if they are included in both graph patterns of a transformation rule.

In the scope of BIM models, the preservation morphism can be understood as the context, new information should be embedded into or removed from. A practical example can be found in the insertion of a physical element like a wall into a building storey, which provides a (local) coordinate system the wall should be placed into. In this case, the coordinate system and the nodes representing the storey are already contained in the host graph the patch is applied to on the receiving side. Accordingly, the transformation rule must provide this contextual information and insert appropriate edges connecting existing nodes and the newly inserted graphlet. Similarly, the removal of information must not affect any shared resources other parts of the object network might still refer to.

Considering the above-mentioned aspects, the corresponding transformation rule is assembled by three essential parts:

1. the graphlet of nodes and edges that have to be inserted or removed
2. the context the graphlet is inserted in or removed from
3. the edges that glue the graphlet and the context

In case of an insertion, the Left Hand Side L and the Interface pattern I contain all nodes and edges describing the context. As secondary nodes lack a unique identifier, each secondary node involved in gluing edges must be specified by a path that contains at least one primary node or connection node. The existence of globally unique IDs attached to these nodes types makes the specified pattern match exactly once in the host graph H. Subsequently, the pattern describing the Right Hand Side R contains all nodes and edges of L and I (to *preserve* their existence) and the graphlet to be inserted. Similarly, removal operations follow similar principles as described for an insertion but with slightly different patterns L and R. In this case, L contains the graphlet to be removed along with the context whereas I and R carry the nodes and edges reflecting the context of the removed graphlet.

Based on the generic concepts introduced before, any modification of a BIM model can be described by a combination (series) of rules and can be captured in DPO-based transformation rules. To apply the transformation on the receiver's side to update the (host) graph, a set of fundamental operations needs to be supported by the graph system. These operations include:

- Match a node or a graph pattern (e.g., to find the Left Hand Side L in the host graph H).
- Insert a new node.
- Insert a new graphlet consisting of nodes and edges that connect these nodes.
- Remove a node and remove the dangling edges that previously connected it to nodes that must be preserved.
- Insert an edge.
- Remove an edge.

The specific operations and statements that need to be performed are derived from the DPO rule.

3.5. The application of CYPHER for encoding graph transformations

For encoding graph transformations, multiple options exist, including the use of bespoke languages of graph transformation systems such as Gr.Gen [59]. In this study, we have chosen the graph manipulation language CYPHER that is operating on property graphs. CYPHER was originally developed for the graph database Neo4j, but is now maintained independently through the openCypher project [60]. It is supposed to become the basis for the graph query language (GQL) standard being developed by ISO. However, it is important to note that CYPHER does not only allow to query graphs but also to modify them. As modifications are located through (host graph) patterns, CYPHER becomes a suitable candidate for representing graph transformation rules.

A decent overview about the CYPHER syntax has been provided by Francis et al. [61] including the set operations that are processed with each keyword. CYPHER was developed to query and mutate property graphs and, therefore, supports numerous concepts. As one of the most essential parts of the syntax, it is possible to specify graph patterns. In a pattern statement, round brackets specify nodes (e.g., `(n1:nodeLabel)`) whereas square brackets declare edges (e.g., `-[e1:edgeLabel]->`). Both, nodes and edges, carry a set of labels that are specified after the colon operator (`:`. Multiple labels can be concatenated using the colon operator several times (e.g., `n1:nodeLabel1:nodeLabel2`). Nodes and edges may be specified using a variable like `n1, e1, ...`. This way, these items can be reused on multiple positions inside a single query statement. To describe a topological structure of a pattern, nodes and edges are assembled in a descriptive fashion. Additionally, nodes and edges may bear property sets, which are specified in curly brackets (e.g. `(n1:nodeLabel {propertyName: "propertyValue"} )`.

The transfer of the transformation rules between a sender and a receiver is ultimately implemented by means of the *Java Script Object Notation* (*JSON*). However, other serialization techniques can also be applied. Listing 1 depicts the transformation rule representing the modification illustrated in figure 2.

```
"structuralModification": {
"L": "(n1:a)-[e1:1]->(n2:b)<-[e2:2]-(n3:c)",
"I": "(n1:a)-[e1:1]->(n2:b), (n3:c)",
"R": "(n1:a)-[e1:1]->(n2:b), (n1)<-[e3:3]-(n3:c)"
 }
```

Listing 1: Transfer of the structural modification using descriptive *CYPHER* statements for the Left Hand Side L, the interface I, and the Right Hand Side R following the Double-Push-Out-Approach. The variables $n1$, $n2$, and $n3$ specify labeled nodes (a, b, c). The variables $e1$, $e2$, and $e3$ define the edges between the nodes.

To apply the transformation rule on a host graph, listing 2 shows how the descriptive statements specified in listing 1 are transformed into a series of CYPHER statements to be performed on the host graph stored at the receiver's machine. Lines starting with `"//"` provide additional comments and are not executed by the statement interpreter.

```
// find left hand side and declare query variables for further use
MATCH (n1:a)-[e1:e1]->(n2:b)<-[e2:e2]-(n3:c)
// remove edge e2 because it was included in L but not in I
DELETE e2
// insert new edge labeled as e3 because it was included in R but not in I
MERGE (n1)<-[e3:e3]-(n3)
```

Listing 2: Application of DPO rule given in listing 1

3.6. Optional back-translation into file-based representations

In order to seamlessly integrate the resulting graph in existing BIM workflows again, it might be necessary to translate the altered graph back into a file-based representation. Depending on the underlying MOF M2 model, the translation of an edge into an association between two entity instances may require the instantiation of identifiers specific to the file (e.g., the entity numbering in STEP files). In the example illustrated in figure 4, the identifier #3 was assigned to a `Point` instance, which in turn got associated with both `Line` instances identified by #1 and #6. These identifiers are only valid locally within the file and can thus change with each serialization process depending on the order the graph nodes are translated into instances. Therefore, the application of the transformation is considered successful if the following criteria are met:

- The resulting graph G_{res} produced by applying the transformation rule p on the outdated graph representation G_{init} is homomorphic to the graph reflecting the updated version $G_{updated}$.
- Commutativity must be given such that the execution of the diff algorithm between G_{init} and $G_{updated}$ results in the same transformation as $diff(G_{init}, G_{res})$.
- According to Kniemeyer [54], DPO-based graph transformations are associative. Hence, the reverse application of the transformation rule p (denoted as p^{-1}) must result in the graph G_A reflecting the state of the initial model version.

3.7. Illustrative example

Figure 7 depicts a situation where a new building element colored in light green has been inserted into a BIM model, which had already contained one building element named *Cuboid1* along with various default information (e.g., unit settings). The new component named *Cuboid2* is represented with a extruded geometry in the three-dimensional design space and and is placed relatively to a coordinate system already specified by the construction site node.

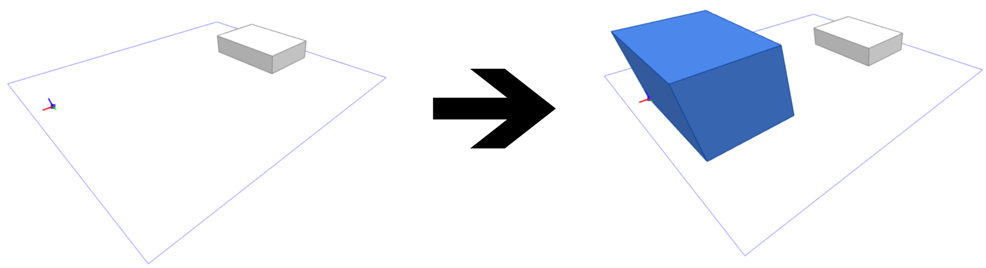

Figure 7: Left: the initial BIM model. Right: the updated version of the BIM model including a newly inserted building component depicted in blue

Following the outlined method, the transformation rule consists of the graphlet to be inserted into the host graph, the context, and the gluing edges that connect the newly inserted graphlet with existing nodes of the host graph. Figure 9 illustrates the nodes to be inserted. Besides, figure 8 shows the nodes and edges forming the context, which the gluing edges connect the push-out graphlet to the existing host graph (depicted in figure 10).

Ultimately, the three parts computed are assembled constituting the Left Hand Side L, the interface I, and the Right Hand Side R. As the modification was an insertion of a new component, the patterns describing L and I consist of the context pattern depicted in figure 8 only. Accordingly, the pattern reflecting R contain the Push Out pattern, the context, and the gluing edges. The entire Right Hand Side pattern is illustrated in figure 11.

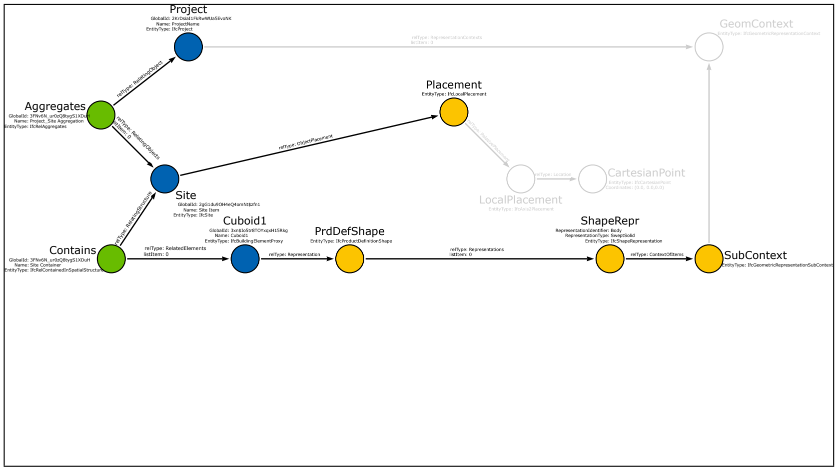

Figure 8: Context pattern

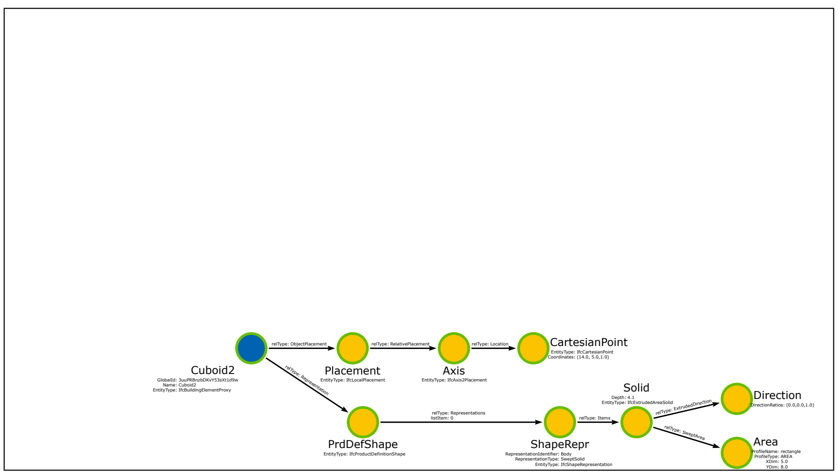

Figure 9: Push Out pattern

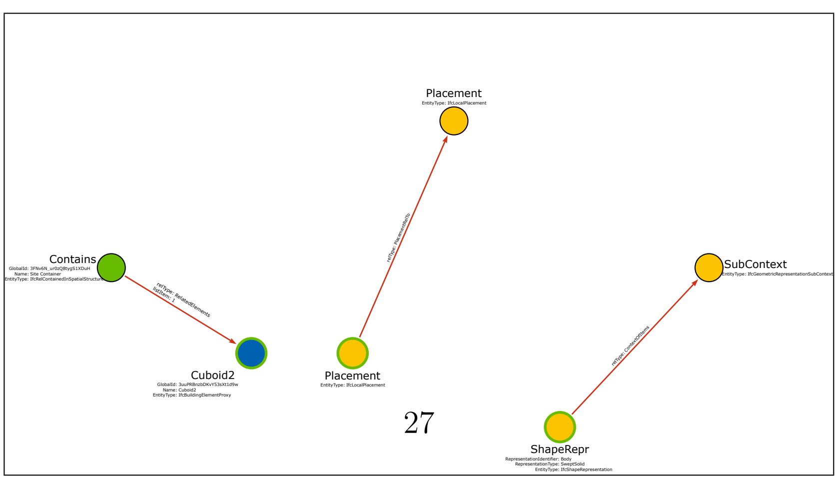

Figure 10: Gluing edges

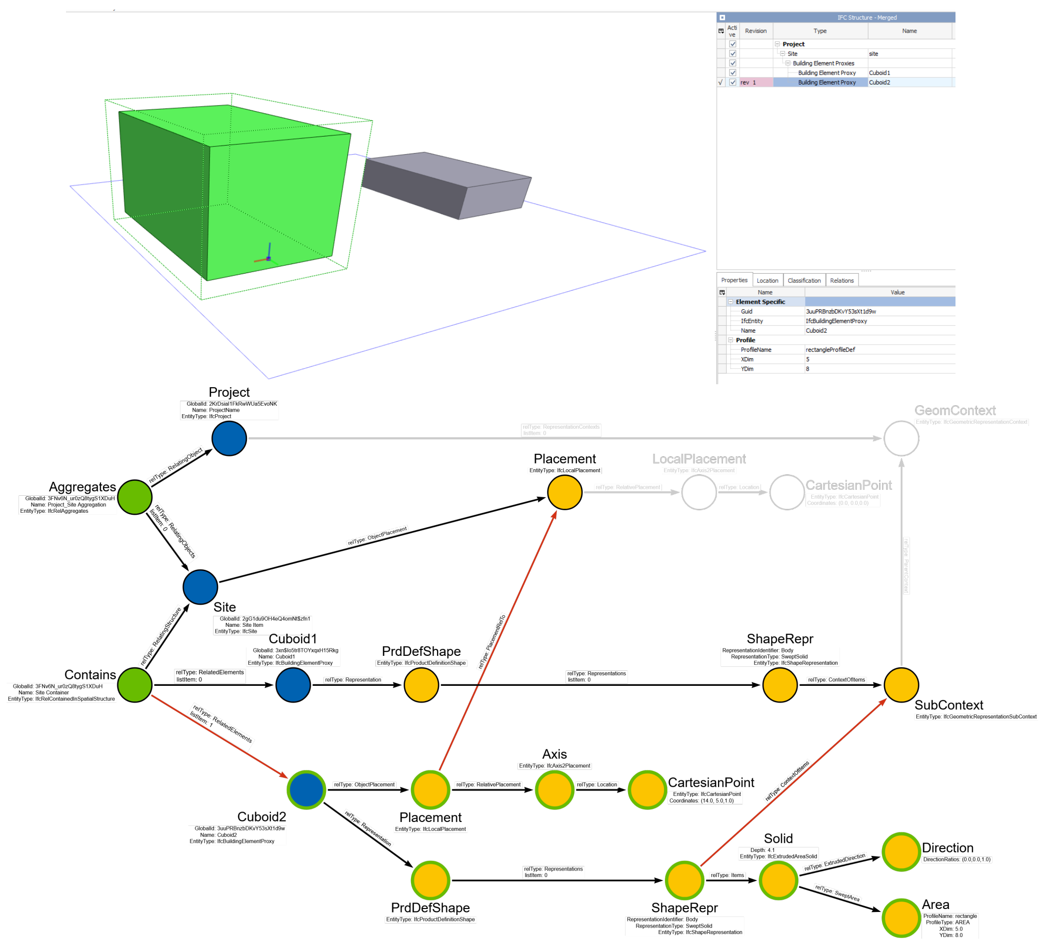

Figure 11: The resulting Right Hand Side pattern R containing all three parts: the push out part, the context pattern and the gluing edges.

4. Case study

The applicability of the proposed method has been tested with several BIM models targeting various use cases. As the underlying data model for validation, the Industry Foundation Classes (IFC) data model developed and maintained by the buildingSMART organization has been chosen. IFC was selected over other established data specifications capturing domain-specific engineering knowledge because of its openness, clear documentation, and broad adoption in the AEC industry and its advanced support in various software applications. In addition to IFC, the data model of LandXML [62] and CityGML [63] were implemented to verify the applicability of the chosen graph meta model independent of the applied data model.

4.1. Experiment setup

The graph database system neo4j by neo technologies [64] was chosen to store and interact with all graph representations produced in the experiment. This graph database is broadly used in industry and academia and supports attributed, multi-labeled graphs with directions. Additionally, various packages for a large set of programming languages and an Application Programming Interface (API) with direct access to the storage system assists in extending core functionalities for specific problems. As neo4j focuses on the storage of graph structures, its support of formal graph transformations implementing Double-Push-Out or Single-Push-Out algorithms is limited. Therefore, a parser was developed that (1) applies the transformation rules to a given graph in the database and (2) translates instance models into their respective graph representations. To test the implemented procedures, comparisons with Gr.Gen has been carried out as well.

Figure 12 visualizes the system setup for the experiment.

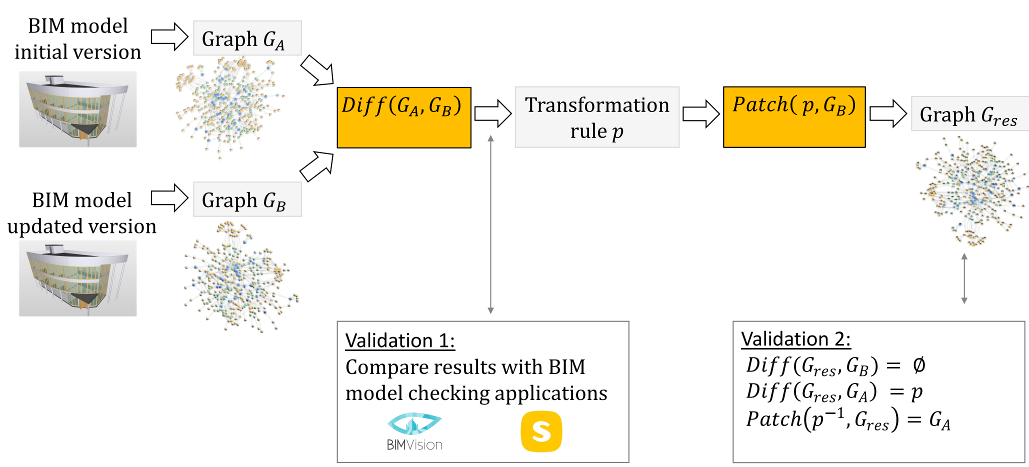

Figure 12: Experiment setup of the case study. All graph representations are stored and managed by a neo4j database. The BIM models implement the IFC data model.

4.2. Result validation

To check and validate the resulting graph structures after applying the transformation rules, two major validation steps have been conducted during the process. Validation 1 compares the results of the *diff* algorithm against the results of BIM model checking tools that provide comparison methods. Here, the well-established software packages Solibri Office [65] and BIM Vision [66] have been used. Additionally, validation 2 aims to proof the correct application of the update patch on a receiving machine according to the criteria specified in 3.4.

4.3. Translation of IFC instance data into the proposed graph structure

For each data model that is supposed to be reflected in the graph structure, a mapping between the corresponding data model and the introduced node types is necessary. For IFC, the publicly available specification defines all IFC classes and is consequently used to map each class to a suitable node type defined in the graph meta model [67]. In general, IFC defines multiple layers in its documentation containing various concept and class definitions. Most important for the mapping are the classes derived from `IfcRoot` and all definitions contained in the resource layer.

The abstract base class `IfcRoot` introduces an *GlobalId* attribute, which serves as a unique identifier for the instances of all sub-classes of this class. From `IfcRoot`, the classes `IfcObjectDefinition`, `IfcRelationship` and `IfcPropertyDefinition` are derived. All instances, which inherit from `IfcObjectDefinition` and `IfcPropertyDefinition`, are handled as `primary nodes` in the graph structure. Entities derived from `IfcRelationship` express one-to-many or many-to-many relationships among instances and are subsequently reflected by the node type `connection node`. Contrary to these uniquely addressable class instances, all classes contained in the resource layer are reflected as `secondary nodes` as these instances are not derived from `IfcRoot`, hence do not carry a unique identifier and might be associated with multiple instances. Figure 13 depicts the mapping of the IFC classes onto the graph meta model.

4.4. Proof of concept

The proof of concept aims to demonstrate the entire process of comparing two model versions to calculate the Delta, formulating the graph transformation rules, and applying them on a receiver's updated graph representation. After multiple tests with IFC-based BIM models, the authors consider the model illustrated in Figure 14 to be representative for typical modifications in AEC design revision processes.

Both versions of the model were created using the authoring software Graphisoft ArchiCAD and exported into the IFC format, version 2x3. The

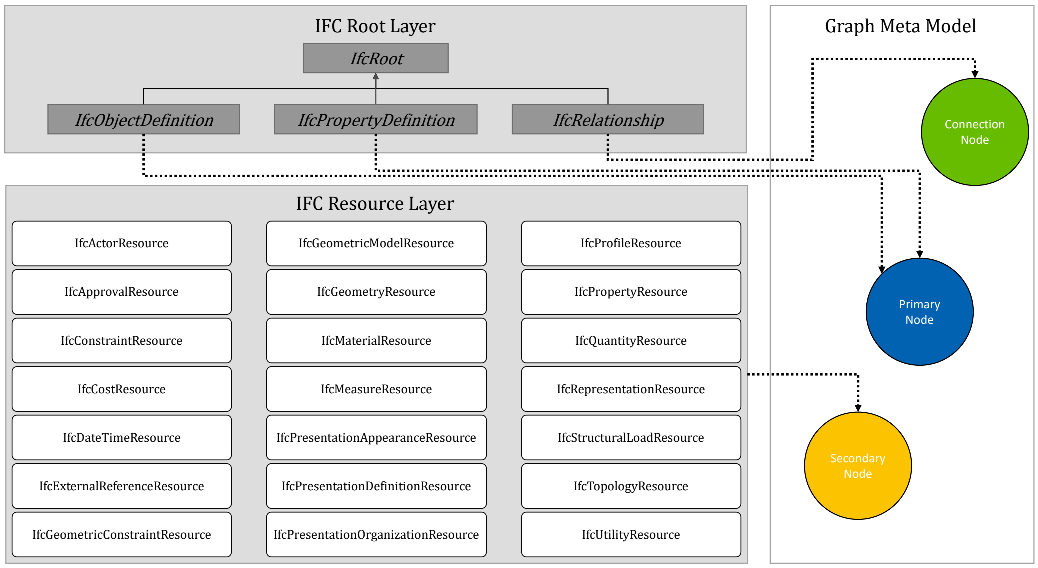

Figure 13: The mapping between the IFC data model and the presented graph meta model illustrated by dashed arrows. Contrary to the classes depicted in the IFC Root Layer, the resource layer subsumes multiple classes in each resource, which are not visualized in this figure.

right rendering in figure 14 depicts the changes applied to the original model. The modifications comprise removing and inserting elements, as well as corresponding changes in the relationship structure and changes in the values of attributes, including the location of elements.

4.4.1. Diff calculation

To detect the modifications between the two versions, the *diff* algorithm presented in section 3.3 was applied. Three model components were detected as *removed*: A wall element in the roof level, a window on the first floor, and an instance of `IfcOpeningElement` entity. Five elements (a door, a wall, and a new ceiling element together with a new space definition and an opening element) were identified as *inserted* in the updated model. Besides removed and inserted components, 27 elements were identified as altered, which has caused 61 node properties to be adjusted. These property modifications reach from updates applied to `PrimaryNodes` (e.g., the modification of the *name* attribute) to changes in the properties of `SecondaryNodes`. All detected changes in the diff computation comply with the modifications identified in

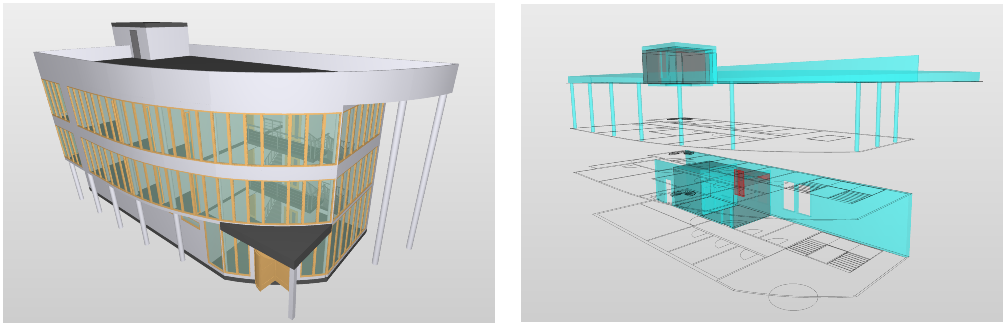

Figure 14: The entire model (left) was tested in the scope of the case study together with a visualization of the applied modifications (right). The visualizations were created using Solibri Office.

the software tools introduced in the system setup.

4.4.2. Formalization of applied modifications in graph transformation rules

As discussed in paragraph 3.4, structural and property modifications are expressed by means of graph transformation rules. As *property modifications* capture attribute updates on nodes that exist in the initial and the updated model, the graphlets defining the Left Hand Side, the interface, and Right Hand Side share the same topological structure and only alter the properties of one node. A typical example of such a property modification is the change of an object placement by updating the Cartesian point defining the location. In the scenario represented in figure 15, the modeler moved an existing wall by a given value in the x-direction. By adding the primary node (the wall) carrying the *GlobalId* property to the patterns, exactly one graphlet of the host graph (i.e., the graph reflecting the initial model version) matches the specified pattern in the Left Hand Side L. Thus, the illustrated patterns replace the *Coordinates* property in node $n4$ with the updated value specified in the Right Hand Side R.

Structural modifications that alter the structure of the graph by inserting or removing nodes and edges require more comprehensive graph transformation rules. In these cases, the transformation rule comprises the graphlet to be removed or inserted and its embedding in the existing graph structure. In the case of inserting a new building component, the graphlet to be inserted may reflect its geometry, the position in the design space, and dependencies to additional semantic information like materials and cost items. Especially, the placement of elements within a logical unit such as a building or a space

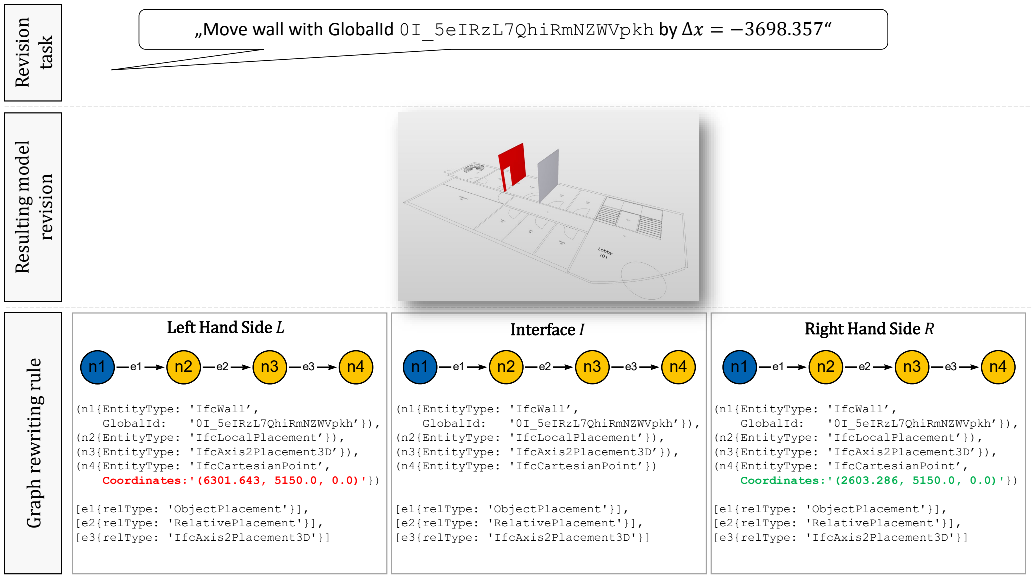

Figure 15: Correlation between a modeling operation and the corresponding graph transformation rule consisting of the Left Hand Side L, Interface I, and the Right Hand Side R. The property sets attached to each node and edge is depicted along with the graph pattern. The specified pattern should be detected exactly once in the host graph because of the given specified *GlobalId* property attached to node *n1*.

is often expressed relative to existing elements. Hence, the inserted graphlet must establish additional edges in the resulting graph to reflect such associations properly. The same principle applies for detected remove operations accordingly. The pattern to be removed from the initial graph must be limited to the nodes that model the actual building component or logical unit whereas its embedding is specified by nodes and edges that are given in the interface pattern I and the Left Hand Side of the rule L.

In addition to removing or inserting operations of entire building components, design refinements to applied to existing components can also occur. These cases appear for example if the geometric representation of a component has been modified to a higher detailing level or expressed by another shape. In these cases, the Left Hand Side specifies the geometric shape to be replaced. The interface pattern I in turn represents the intermediate state, in which the component only exists in its semantic skeleton without any geometry. Finally, the Right Hand Side R describes the new geometric shape to be inserted and leads to the updated version.

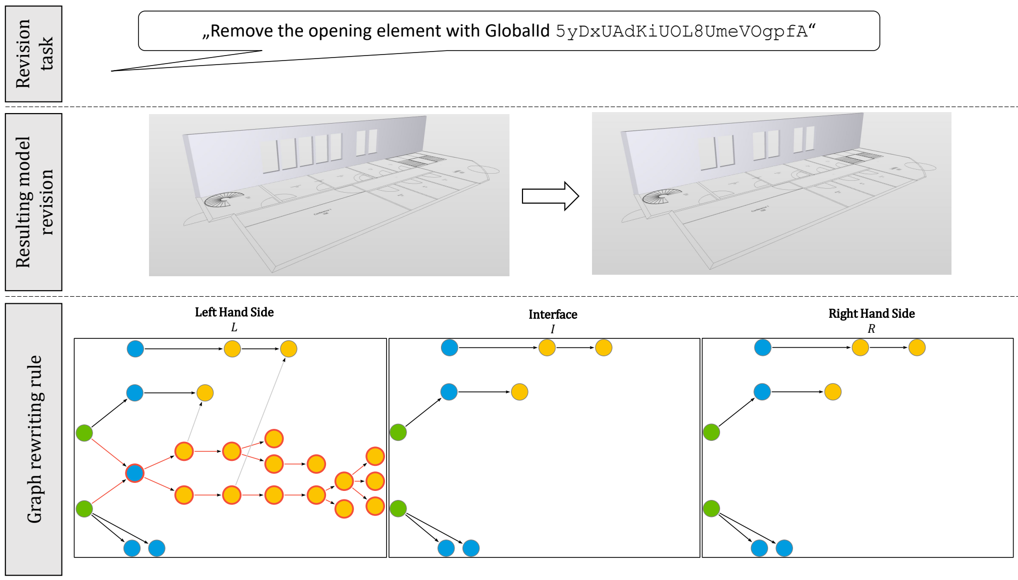

Figure 16: Correlation between applied modification, resulting BIM model, and the formulation of the corresponding graph rewriting rule following the DPO approach.

Figure 16 illustrates the correlation between the editing task of removing an opening element, the BIM model transformation, and the graph-based rewriting rule. Nodes depicted with red bounds represent the opening element and are identified as nodes to be removed (i.e., the push out part of the transformation rule). Edges depicted in gray represent gluing edges between the removed graphlet and parts of the graph that must remain unchanged during the transformation. Hence, only the edge itself, but not the target node is removed from the graph. Because `secondary nodes` lack a unique property, the associated resources must be specified by a graph pattern that includes a `primary node` as "anchor", which provides the context of the transformation rule. By utilizing the feature of globally unique identifiers assigned to each primary node, each secondary node that has an inbound gluing edge is uniquely specified, which ensures the correct application of the transformation rule on the receiver's system.

Tables 1 and 2 provide a quantitative insight into the number of nodes included in each rewriting rule reflecting the removal or insertion of a model component.

In addition to structural modifications, the system reports on 60 modified node attributes affecting 29 model components. The average path length

Removed components	removed nodes	context secondary	context primary
Opening element	18	6	5
Window	374	10	109
Wall	24	10	49

Table 1: Quantitative assessment of removed components in the BIM model

Inserted components	inserted nodes	context secondary	context primary
Door	71	6	5
Wall	377	10	107
Covering	53	13	142
Space	48	9	93
DoorType	48	9	60

Table 2: Quantitative assessment of inserted components in the BIM model

between the modified node and its parent primary node is 4.9 where the minimum distance is 0 (e.g., the modification of the *name* attribute attached to a primary node) and the maximum distance is 8. Together with the necessary patterns for inserting and removing model components and their associated resources, the graph transformation rules are composed of 3,828 nodes in total. Each model version has resulted in around 186,700 nodes reflecting one model version. Comparing the results of the presented experiment with the total number of nodes per model, our patch-based mechanism demands only 2% of the nodes reflecting an entire model to properly describe the applied modification.

The transfer of the formulated transformation rules was realized using the data structure outlined in 3.4. According to the quantitative assessments in table 1 and 2, the patterns used in these transformation rules are much larger than the example illustrated in 3.7 and therefore difficult to visualize in their full extend.

4.4.3. Results

The proposed version control system detected all expected differences between the tested model versions, which provided all necessary information to formulate suitable graph transformation rules. Furthermore, the system was capable of describing all detected modifications in suitable transformation rules that have been ultimately applied to the initial version. The comparison of the transformed graph with the graph produced by the updated model file has resulted in no differences, which has proven the successful transfer and application of the transformation rules.

5. Discussion

The approach discussed in this paper builds upon an entity-centered approach to reflect the object network of a BIM model in an attributed, directed graph and transfers updates using formal graph transformation rules. The proposed method lays the foundation for tracking updates applied to a BIM model on object basis instead of deploying entire files containing the updated design information. Incremental updates support the precise tracking of changes and help engineers to focus on the impact modifications in "foreign" models may have on their own discipline models. Additionally, the proposed method creates new possibilities for automated (yet controlled) updates across disciplines wherever a formal description of dependencies across domains is possible. Nevertheless, the principle of discipline-oriented ownership and responsibility remains fully respected.

The method is generic in the sense that it is independent of the underlying data model. The prerequisites are limited to the existence of a well-defined data model that employs the basic principles of object-oriented data modeling. Furthermore, the proposed *diff* computation is based on the existence of identifiers, which are considered as consistent across all versions. The formulation of patches utilizes well-defined concepts of graph transformations and is backed up by numerous research contributions and algebraic proofs. A new contribution in this regard is the reverse construction of the transformation rule by comparing the initial and the final state of a model and the integration of common characteristics of the AEC industry. Especially the abstract concept of preservation morphisms have been set into a practical context and will provide the basis for future research and developments. The system fits perfectly into existing applications and BIM workflows, which is assured by providing a one-to-one bidirectional translation between the instances populated in a BIM model and its corresponding graph representation.

5.1. Algorithmic limitations

Despite the successful test of the developed version management system in the case study, some challenges remain unsolved and require further investigation. One major limitation lies in the traversing nature of the *diff* computation. By storing node pairs that have been already identified as equivalent, the algorithm traverses each pair only once. At the same time, each node is visited at least once. Hence, the computation cannot be executed in concurrent threads, which would reduce the computational time needed to traverse the entire graphs of both versions compared. A possible improvement can be seen by shrinking the graph to nodes that reflect only semantic information of a model and to outsource geometric representations into external storage types. Such modification in the translation of models into their graph representation, however, breaks the concept of reflecting each entity instance in the BIM model by exactly one node in its graph. This condition greatly assists in developing new interfaces for additional data models as no special treatment for specific domain representations is required. Rather, it just employs the object-oriented principles of classes and associations.

In addition to the limitations caused by the traversing strategy, future improvements are necessary to detect advanced modifications such as the re-assembly of components within the logical structure of a model or refinement actions. As outlined in the motivation, BIM models typically evolve over time, which often leads to scenarios where a single component is replaced by an assembly of elements with higher detailing. The current method identifies such modifications as a combination of removals and insertions and will deploy both transformations accordingly. Of course, the object-based synchronization of all replicas can be achieved successfully. From a user perspective, however, the information about the shift of an existing component would be much more valuable to attach discipline-specific routines.

5.2. Domain-specific limitations

In addition to limitations caused by the chosen comparison and transformation strategies, the method relies on the existence of stable identifier attributes such as the *GlobalId* attribute. Even though the definition implies the intention to *identify* an object across several occurrences, current IFC export interfaces of renowned software providers prove that consistent object IDs are not self-evident. Hence, additional investigations are required to identify components among different versions that still reflect the same object but might have an altered identifier. Apart from the *GlobalId* attribute, characteristics like the position of a component of the spatial containment may assist here. Such variations are difficult to solve on a generic level, because knowledge about the specific application field and the reflected data set is required.

Even though the general approach has been tested successfully on BIM models implementing the IFC data standard, some peculiarities specific to data models in the AEC sector hamper a performance-efficient application of the proposed method so far. Especially explicit boundary representations defining the shape of a component require an extensive amount of nodes and edges to reflect all instances contained in the BIM model. Figure 17 depicts a quantitative analysis of the IFC entities mostly instantiated in the models presented in the case study. Instances of the `IfcCartesianPoint` entity cause around one-third of all instances included in the BIM model. Some of them specify the placement of components within the model. However, the dominant majority of these nodes are instantiated to model explicit boundary representations of components.

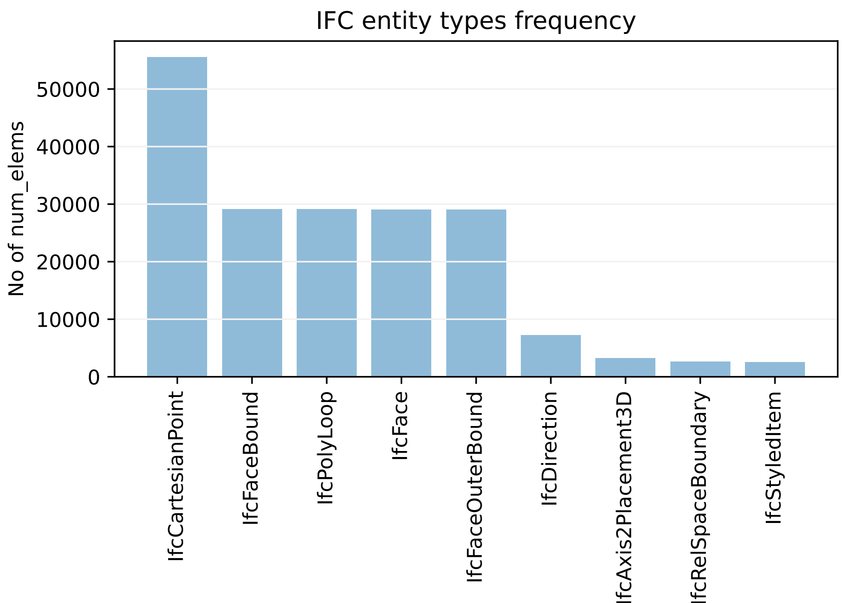

Figure 17: Frequency plot of the entities instantiated in the building model presented in the experiment. The initial and the revised version of the BIM model result in a very similar frequency plot. Therefore, only the quantitative analysis of the initial version is depicted in this figure.

The current system does not provide any schema preservation mechanisms so far. It is theoretically possible to alter graph representations in a way that they do not comply with the restrictions defined in the underlying data model. Similar to the issue of modified identifier attributes, this aspect can only be circumvented by working in controlled environments, which prevent the user from modifications that are not schema-compliant. A suitable approach may be given by incorporating preliminary checks of the generated graph transformation rules against compliance rules. Conformance checking of data sets opens another large field with a vast number of existing techniques, definitions, and experiments that will support future improvements in patch-based update strategies.

6. Summary and Outlook

The current practise of BIM-based collaboration is mainly based on the transfer of entire discipline-specific instance models. As changes are tracked on the granularity of entire BIM models stored in files, object-based change tracking and version control are not yet provided in today's CDE systems. This lack requires the manual identification of changes applied to new model versions.

To overcome the addressed limitations, the paper at hand contributes a generic approach that is applicable to instances of any object-oriented data model. The core of the approach lies in representing BIM models as graphs and employing the concept of graph transformations to describe the modifications applied to the BIM model. Hence, only these modifications are transferred by sending the graph transformations to the receiver. Modifications are formally described by means of transformation rules implementing the well-established concept of Double-Push-Out rewriting. The concept can be applied in different distributed system setups, including client-server systems with a central repository as well as dispersed peer-to-peer networks.

Regarding the collaborative process, the stakeholders of any discipline continue to work in an asynchronous collaboration mode, using the design environment of their choice. They keep the responsibility over their authored discipline models as demanded by ISO 19650. In addition, each designer has full control over when a model should be made available to other project participants. Once a new version of a BIM model is authored and ready to be shared with other project partners, the version management system identifies the applied modification between the initial and the updated state of the BIM model. Thereafter, only this modification is transmitted by means of graph transformations. On the receiver's machine, the application of the incoming patch modifies the graph reflecting the outdated model version to the most recent state, which leads to a consistent up-to-date representation of the BIM model at all involved parties.

The proposed system becomes particularly useful for any design project that involves a multitude of disciplines, experts, and subsequently large model files. Especially in complex design tasks that require the expertise of many experts, the patch-based exchange of new model versions can significantly enhance the overall collaboration workflow by providing quick access to new versions and the ability to perform subsequent processing of incoming modifications. This way, engineers and designers can directly evaluate the impact of modifications in foreign discipline models on their specific design tasks and may alter their own models accordingly.

In the future, the management of large geometric representations requires dedicated emphasis and should aim to overcome the limitations addressed. Nonetheless, the proposed method reveals great potential by combining established principles of BIM-based collaboration with formal graph theory and object-based model synchronization.

Acknowledgments

We gratefully acknowledge the support of the German Research Foundation (DFG) for partly funding the project under grant FOR2363. Additionally, we would like to thank Autodesk, Inc. for their financial support.

References

[1] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors, John Wiley & Sons, Inc., 2008.

[2] A. Bradley, H. Li, R. Lark, S. Dunn, BIM for infrastructure: An overall review and constructor perspective, Automation in Construction 71 (2016) 139–152.

[3] A. Borrmann, M. König, C. Koch, J. Beetz, Building Information Modeling: Why? What? How?, in: Building Information Modeling, Springer International Publishing, Cham, 2018, pp. 1–24.

[4] ISO, Iso 19650-1 organization and digitization of information about buildings and civil engineering works, including building information modelling (bim)–information management using building part 1: concepts and principles information modelling, 2018.

[5] C. Preidel, A. Borrmann, H. Mattern, M. König, S.-E. Schapke, Common Data Environment, in: Building Information Modeling, Springer International Publishing, Cham, 2018, pp. 279–291.

[6] J. Radl, J. Kaiser, Benefits of Implementation of Common Data Environment (CDE) into Construction Projects, IOP Conference Series: Materials Science and Engineering 471 (2019).

[7] M. Taylor, Crossrail project: building a virtual version of London's Elizabeth line, Proceedings of the Institution of Civil Engineers - Civil Engineering 170 (2017) 56–63.

[8] M. Oh, J. Lee, S. W. Hong, Y. Jeong, Integrated system for bim-based collaborative design, Automation in Construction 58 (2015) 196–206.

[9] Object Management Group, OMG Meta Object Facility (MOF) Core Specification, 2019.

[10] J. F. Overbeek, Meta Object Facility (MOF) investigation of the state of the art, 2006.

[11] P.-H. Chen, L. Cui, C. Wan, Q. Yang, S. K. Ting, R. L. Tiong, Implementation of ifc-based web server for collaborative building design between architects and structural engineers, Automation in Construction 14 (2005) 115–128.

[12] BSi, Pas 1192-2: 2013: Specification for information management for the capital/delivery phase of construction projects using building information modelling, 2013.

[13] S. Chacon, Pro Git, Apress, 2009.

[14] J. D. Blischak, E. R. Davenport, G. Wilson, A Quick Introduction to Version Control with Git and GitHub, PLoS Computational Biology 12 (2016) 1–18.

[15] C. Koch, B. Firmenich, An approach to distributed building modeling on the basis of versions and changes, Advanced Engineering Informatics 25 (2011) 297–310.

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

[17] Autodesk, Revit cloud worksharing — autodesk bim 360, 2021. Https://www.autodesk.de/bim-360/design-collaboration/revit-cloud-worksharing/ (visited on 2021-12-10).

[18] Tekla, Tekla model sharing - bim-basierte zusammenarbeit — tekla, 2021. Https://www.tekla.com/de/produkte/tekla-model-sharing (visited on 2021-12-10).

[19] GRAPHISOFT, Bimcloud bim without constraints, 2021. Http://www.graphisoft.com/bimcloud/overview/ (visited on 2021-12-10).

[20] S. Boeykens, Bridging building information modeling and parametric design, in: eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2012, Taylor and Francis Group, 2012, pp. 453–458.

[21] P. Poinet, D. Stefanescu, E. Papadonikolaki, Collaborative Workflows and Version Control Through Open-Source and Distributed Common Data Environment, volume 98, Springer International Publishing, 2020.

[22] M. Chein, M.-L. Mugnier, M. Croitoru, Visual reasoning with graph-based mechanisms: the good, the better and the best, The Knowledge Engineering Review 28 (2013) 249–271.

[23] A. Kneidl, A. Borrmann, D. Hartmann, Generation and use of sparse navigation graphs for microscopic pedestrian simulation models, Advanced Engineering Informatics 26 (2012) 669–680.

[24] B. Helms, K. Shea, Computational synthesis of product architectures based on object-oriented graph grammars, Journal of Mechanical Design 134 (2012).

[25] S. Kwon, L. V. Monnier, R. Barbau, W. Z. Bernstein, Enriching standards-based digital thread by fusing as-designed and as-inspected data using knowledge graphs, Advanced Engineering Informatics 46 (2020).

[26] J. Hao, L. Zhao, J. Milisavljevic-Syed, Z. Ming, Integrating and navigating engineering design decision-related knowledge using decision knowledge graph, Advanced Engineering Informatics 50 (2021).

[27] J. Johansson, M. Contero, P. Company, F. Elgh, Supporting connectivism in knowledge based engineering with graph theory, filtering techniques and model quality assurance, Advanced Engineering Informatics 38 (2018) 252–263.

[28] A. Singh, R. Brennan, D. O'Sullivan, DELTA-LD: A change detection approach for linked datasets, 4th Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW) (2018).

[29] A. Braun, S. Tuttas, A. Borrmann, U. Stilla, Automated progress monitoring based on photogrammetric point clouds and precedence relationship graphs, in: Proceedings of the International Symposium on Automation and Robotics in Construction, IAARC Publications, 2015, pp. 1–7.

[30] E. Tauscher, H.-J. Bargstädt, K. Smarsly, 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, pp. 905–912.

[31] V. J. Gan, Bim-based graph data model for automatic generative design of modular buildings, Automation in Construction 134 (2022).

[32] B. Strug, G. lusarczyk, A. Paszyska, W. Palacz, A Survey of Different Graph Structures Used in Modeling Design, Engineering and Computer Science Problems, volume 107, Springer Science and Business Media B.V., pp. 243–275.

[33] E. Curry, J. ODonnell, E. Corry, S. Hasan, M. Keane, S. ORiain, Linking building data in the cloud: Integrating cross-domain building data using linked data, Advanced Engineering Informatics 27 (2013) 206–219.

[34] M. H. Rasmussen, M. Lefranois, P. Pauwels, C. A. Hviid, J. Karlshj, Managing interrelated project information in aec knowledge graphs, Automation in Construction 108 (2019) 102956.

[35] J. Beetz, J. Van Leeuwen, B. De Vries, Ifcowl: A case of transforming express schemas into ontologies, Ai Edam 23 (2009) 89–101.

[36] P. Pauwels, T. Krijnen, W. Terkaj, J. Beetz, Enhancing the ifcowl ontology with an alternative representation for geometric data, Automation in Construction 80 (2017) 77–94.

[37] C. Zhang, J. Beetz, B. de Vries, Bimsparql: Domain-specific functional sparql extensions for querying rdf building data, Semantic Web 9 (2018) 829–855.

[38] J. Oraskari, S. Törmä, RDF-based signature algorithms for computing differences of IFC models, Automation in Construction 57 (2015) 213–221.

[39] M. H. Rasmussen, M. Lefranois, M. Bonduel, C. A. Hviid, J. Karlshj, Opm: An ontology for describing properties that evolve over time, in: Proceedings of the 6th Linked Data in Architecture and Construction Workshop, pp. 24–33.

[40] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, Y. Stavrakas, A Flexible Framework for Understanding the Dynamics of Evolving RDF Datasets, in: M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. D'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, K. Thirunarayan, S. Staab (Eds.), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 9366 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, 2015, pp. 495–512.

[41] C. Bobed, P. Maillot, P. Cellier, S. Ferré, Data-driven assessment of structural evolution of RDF graphs, Semantic Web 11 (2020) 831–853.

[42] Q. Zhao, Y. Li, X. Hei, M. Yang, A Graph-Based Method for IFC Data Merging, Advances in Civil Engineering 2020 (2020) 1–15.

[43] X. Shi, Y. S. Liu, G. Gao, M. Gu, H. Li, IFCdiff: A content-based automatic comparison approach for IFC files, Automation in Construction 86 (2018) 53–68.

[44] A. Schultheiß, A. Boll, T. Kehrer, Comparison of Graph-based Model Transformation Rules., The Journal of Object Technology 19 (2020) 3:1.

[45] Y. Wang, C. Maple, A novel efficient algorithm for determining maximum common subgraphs, Proceedings of the International Conference on Information Visualisation 2005 (2005) 657–663.

[46] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, M. Vento, A comparison of algorithms for maximum common subgraph on randomly connected graphs, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2396 (2002) 123–132.

[47] D. Conte, P. Foggia, M. Vento, Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs, Journal of Graph Algorithms and Applications 11 (2007) 99–143.

[48] G. Rozenberg, Handbook of graph grammars and computing by graph transformation, volume 1, World scientific, 1997.

[49] J. Blomer, R. Geiß, E. Jakumeit, The GrGen.NET User Manual, 2013.

[50] H. Ehrig, M. Pfender, H. J. Schneider, Graph-grammars: An algebraic approach, 14th Annual Symposium on Switching and Automata Theory (1973) 167–180.

[51] R. Geiß, G. V. Batz, D. Grund, S. Hack, A. Szalkowski, GrGen: A Fast SPO-Based Graph Rewriting Tool, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 4178 LNCS, 2006, pp. 383–397.

[52] P. M. van den Broek, Algebraic graph rewriting using a single pushout, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 493 LNCS (1991) 90–102.

[53] S. Buchwald, Erweiterung von grgen.net um dpo-semantik und ungerichtete kanten, 2008.

[54] O. G. M. Kniemeyer, Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling, Ph.D. thesis, Brandenburgische Technische Universität Cottbus, 2008.

[55] M. Javed, Operational change management and change pattern identification for ontology evolution, 2013.

[56] J. Abualdenien, A. Borrmann, Pbg : A parametric building graph capturing and transferring detailing patterns of building models, in: Proc. of the CIB W78 Conference, pp. 11–15.

[57] C. Königseder, A Methodology for Supporting Design Grammar Development and Application in Computational Design Synthesis, Ph.D. thesis, ETH Zurich, 2015.

[58] J. Hidders, A Graph-based Update Language for Object-Oriented Data Models, Ph.D. thesis, Eindhoven University of Technology, 2001.

[59] E. Jakumeit, S. Buchwald, M. Kroll, Grgen.net, International Journal on Software Tools for Technology Transfer 12 (2010) 263–271.

[60] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 1433–1445.

[61] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, M. Schuster, P. Selmer, A. Taylor, Formal semantics of the language cypher, arXiv preprint (2018).

[62] LandXML.org, Landxml, 2022. Http://www.landxml.org/ (visited on 2022-01-05).

[63] G. Gröger, T. H. Kolbe, C. Nagel, K.-H. Häfele, OGC city geography markup language (CityGML) encoding standard, Open Geospatial Consortium, 2.0.0 edition, 2012.

[64] neo technology, Neo4j, 2022. Https://neo4j.com// (visited on 2022-01-05).

[65] Solibri, Solibri office, 2022. Https://www.solibri.com (visited on 2022-04-07).

[66] DataCubist, Bimvision, 2022. Https://bimvision.eu/de/ (visited on 2022-04-07).

[67] ISO, ISO 16739-1:2018: Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries - Part 1: Data schema, 2018.