Automation in Construction 155 (2023) 105063

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Version control for asynchronous BIM collaboration: Model merging through graph analysis and transformation

Sebastian Esser [*], Simon Vilgertshofer, André Borrmann

Chair of Computational Modeling and Simulation, School of Engineering and Design, Technical University of Munich, Germany

ARTICLE INFO

Keywords:
Model synthesis
BIM Level 3
Graph transformation
Patching
Merging
Time-evolving BIM models
Common Data Environment
Model Coordination

ABSTRACT

The design process in construction projects is iterative and multi-disciplinary in nature. In today's industry practice, several discipline experts concurrently author multiple versions and design variants of BIM models and share them at frequent intervals. Applying a sound version control methodology can significantly enhance automation, enabling the coordination and combination of these model versions into consistent overall models with less extensive manual effort. This paper introduces a diff-and-patch mechanism for transferring changes between model versions, facilitating object-level change tracking using graph representations of BIM model data, and specifically focuses on merging diverging versions through the application of graph transformations. The mechanisms for executing branching and merging of model versions are thoroughly explained and showcased through various illustrative scenarios. The presented method adheres to the established principles of federated BIM collaboration but equips the participating parties with additional means to automate the combination of various model versions, allowing them to focus on the relevant conflicts. The proposed methodology of graph-based version control unlocks the potential of analyzing interdisciplinary dependencies across partial models and enables the more efficient resolution of conflicting model versions.

1. Introduction

The AEC industry is characterized by a large number of highly specialized disciplines. Especially in projects with various, sometimes exceptional boundary conditions, many experts must contribute their knowledge to achieve a design solution that fulfills the manifold requirements and constraints inherent in a construction project. This makes building design a highly collaborative process.

Over the last two decades, the technology of Building Information Modeling (BIM) has been increasingly adopted by the AEC industry [1,2]. Hence, the traditional approach of exchanging mainly (printed) documents and drawings has been superseded by creating information-rich BIM models that provide a comprehensive digital representation of the built asset. A BIM model assembles geometric and semantic information about the built asset and can be encoded in vendor-specific (i.e., closed) or vendor-neutral, open data formats [3]. Public authorities demand model deliveries in open, standardized representations to enable fair competition on the software market [4,5].

Various standards and guidelines have been defined on national and international levels to structure the workflows of collaborative design projects in the construction industry. The most relevant one is ISO 19650-1, which has been defined as a successor to the British standard PAS 1192 [6,7]. Both describe process-related aspects and promote BIM-based collaboration using loosely coupled (i.e., federated) discipline models. According to ISO 19650, each domain works predominantly independently of the other disciplines on their respective discipline models [8]. Once a shareable state is reached, the design information is uploaded as a complete model file (also denoted as an information container) to a joint project space known as a Common Data Environment (CDE) [9]. By employing a CDE, interdisciplinary coordination can be performed by combining the available discipline models into a coordination model. From a technical point of view, this combination step of all discipline models is realized by merely superimposing the elements contained in the individual discipline models.

1.1. Problem statement

The notion of federated models and interdisciplinary collaboration based on frequent whole-model transfers averts the tracking of changes

[*] Corresponding author.
E-mail address: sebastian.esser@tum.de (S. Esser).

https://doi.org/10.1016/j.autcon.2023.105063
Received 18 April 2023; Received in revised form 9 August 2023; Accepted 15 August 2023
Available online 28 August 2023

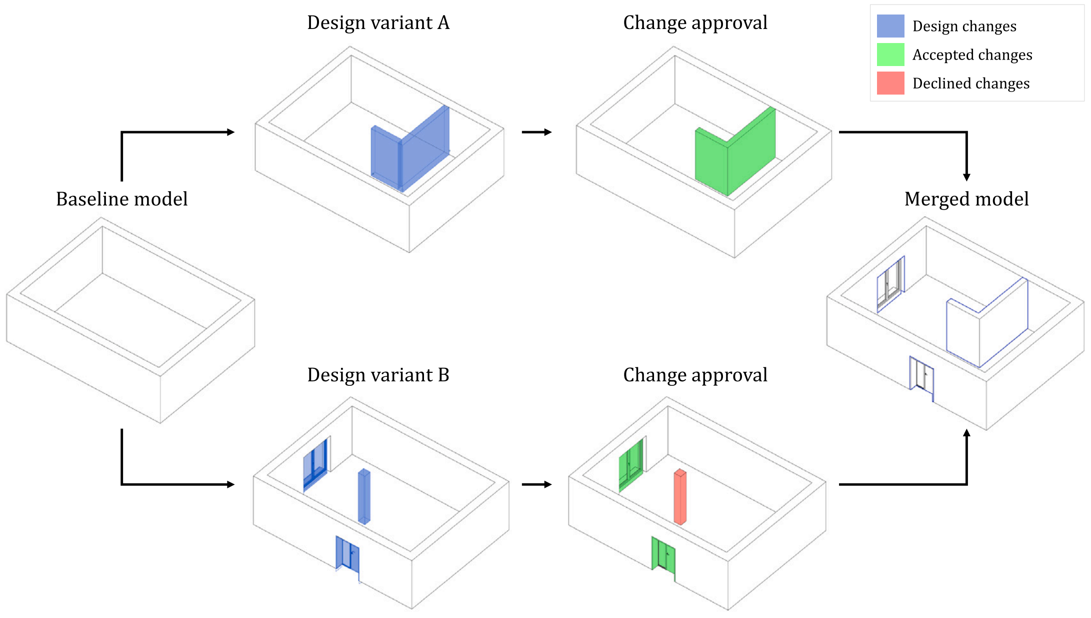


Fig. 1. Multiple diverging design variants of a BIM model produced during a design process. There is no comprehensive mechanism to merge design decisions into a joint deliverable using vendor-neutral data models.

at the level of individual objects stored inside a discipline model. This lack of object-level concurrency control results in a significant drawback: All other stakeholders must manually identify the changes performed by one stakeholder to update their discipline models accordingly. This also means that concurrent changes of the same or related model elements by two different stakeholders may remain unidentified and lead to significant inconsistencies. Given the high complexity of collaborative design, the lack of support for object-level concurrency control, including fine-grained tracking of changes and automated conflict identification during merging, results in an error-prone and laborious manual process of model coordination, leaving the potential of computational support widely untapped.

From a more high-level perspective, it must be stated that the current best practice of design coordination using federated discipline models does not adequately address the nature of highly iterative design processes. Equivalent problems can be seen in the local scope of a single designer participating in a BIM-based collaboration workflow: Multiple variants of BIM models are produced to find the best-performing solution for a given design task. Again, the current best practice is to store all variants as individual files, resulting in redundancy and loss of connection between the common basis of all variants.

To overcome the above-mentioned deficiencies, this paper presents an approach for implementing optimistic concurrency control on the object level for concurrently modified BIM models. The recombination (merging) of concurrently developed models has to take into account (I) non-conflicting parts where the collaborators made modifications on disjoint subsets and (II) conflicting parts where the same objects have been altered, potentially in a non-consistent manner. This paper addresses identifying these two parts and their proper handling regarding concurrent version control. It uses graph representations of BIM models and graph transformations to describe incremental changes on the object level as introduced in Esser et al. [10]. In addition, detailed emphasis is spent on existing technologies for data merging used in other industry branches and why they cannot be applied to data representations used to express built environment assets without further adaptation.

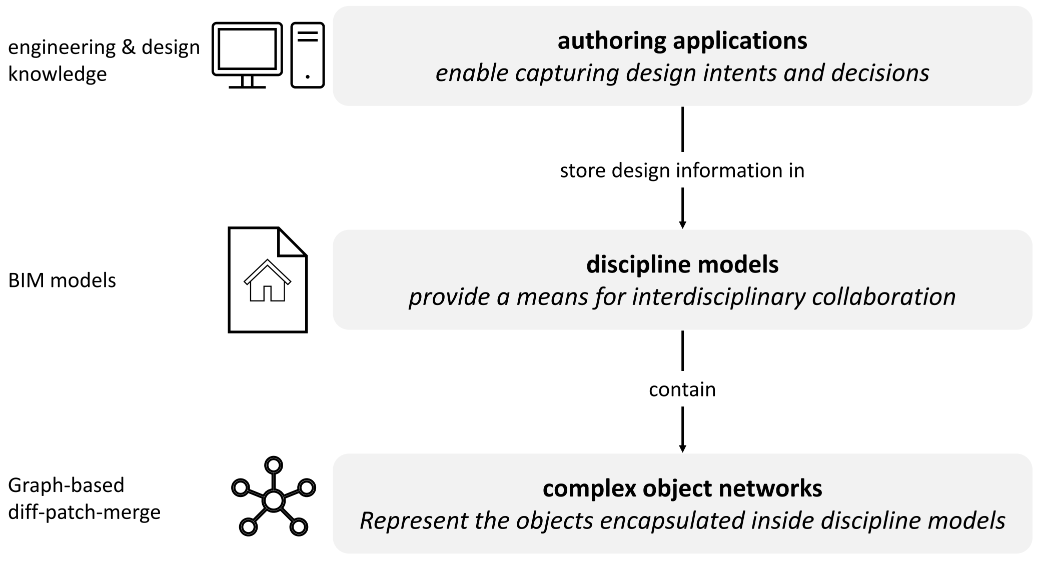
1.2. Scope of the paper

The analysis of the current situation shows that collaboration based on federated discipline-specific models is well established [8,9]. However, it is necessary to improve the issue of repetitive model exchanges and consider the evolving nature of BIM models in the design process. Looking into other industry sectors, established concepts of concurrency control including branching and merging provide a promising solution to overcome the limitations identified.

We will first summarize previous works addressing model-based version control using incremental update patches for the presented problem field. As a critical contribution, we will extend these preliminary achievements by introducing the concepts of *branching* and *merging* to enable concurrent model states. We will define conditions under which the merging of increments reflecting diverging design states of a BIM model can be performed. After a formal definition of the problem with fundamental notions of set theory and graph transformations, several scenarios are outlined from which the necessity of merging diverging model states is motivated. From these scenarios, a general condition evaluation is derived that is agnostic to the data format used to encode the engineering knowledge in a graph.

Introducing abstraction layers to structure the terminology and gain a common understanding of the proposed methods is vital. Fig. 2 illustrates the three elementary layers. As the top priority, engineers and designers involved in collaborative BIM workflows provide rich knowledge about their specific discipline and operate BIM authoring applications. To capture decisions and results, discipline models are employed to store specific design information about the built asset in question. Ultimately, BIM models are assembled by a set of objects that can be understood using complex object networks as their technical foundation.

Unless otherwise mentioned, this paper mainly discusses aspects of the most generic abstraction layer that features the notion of graphs as its fundamental concept. At appropriate positions, however, the conclusions will be set into a reasonable context that relates to practical examples.



engineering & design knowledge
authoring applications
enable capturing design intents and decisions
store design information in
BIM models
discipline models
provide a means for interdisciplinary collaboration
contain
Graph-based diff-patch-merge
complex object networks
Represent the objects encapsulated inside discipline models

Fig. 2. Different abstraction layers are used in the following explanations.

1.3. Contributions & structure

In summary, the main contributions presented in this paper are as follows:

- A proposal for an optimistic concurrency framework that extends preliminary works of incremental model patches by the concepts of *branching* and *merging*.
- The definition of conditions under which diverging model stages can be eventually merged without any conflicts.
- An outline of how the presented approach can be facilitated in future BIM collaboration workflows.

The paper is organized as follows: Section 2 discusses related work in the field, and Section 3 introduces the concept of branching and merging. Furthermore, this section presents the developed conditions that must be met to merge diverging model stages. Next, Section 4 discusses the theoretical approach in practical scenarios and provides an outlook on how the developed concept can be incorporated into future BIM collaboration platforms. Section 5 discusses identified limitations and Section 6 concludes the paper with a summary of the main findings.

2. Background & related work

Collaborative approaches in which a system eventually allows diverging states of a digital artifact have been the subject of research in various fields [11]. Therefore, this chapter aims to introduce essential aspects of *transactions* in computer systems that form the technical foundation for all subsequent considerations. Additionally, existing approaches for incremental version control techniques in object networks and BIM collaboration will be summarized. Lastly, some technical background information regarding graph transformations and their commutativity is presented.

2.1. Transactions on computer systems

A transaction is commonly defined as an atomic unit of actions that transforms an object from one valid state into another valid state. Typically, a transaction consists of a set of operations. The transaction is committed only if all operations can be performed successfully. In the case of failures during the execution of a transaction, all operations are rolled back to ensure a valid object state at any time [12]. Transactions are widely implemented to preserve a system in a consistent state and are therefore often associated with concurrency control. Especially in database applications, transactions can be further described by the *ACID* principles. According to Gray [12] and Robinson et al. [13], the abbreviation *ACID* represents the concepts of *Atomicity*, *Consistency*, *Isolation*, and *Durability*.

Even though these principles are still widely used in many computer systems, various application cases have been identified that neither require immediate consistency nor fit a strictly pessimistic concurrency control model. To soften the requirements of immediate consistency, the *BASE* principles have been defined as a more optimistic concurrency paradigm:

- Basic Availability: The stored data is not blocked during transactions. Thus, data is available most of the time.
- Soft state: Stored operations do not have to be consistent during the writing operation and do not directly federate to all existing replicas of the data set.
- Eventual Consistency: Consistency among replicas is achieved eventually at a later point in time.

According to these elemental concepts of storing and manipulating information in a computer system, it becomes evident that the established fashion of decoupled discipline models in BIM-based collaboration associates well with the described *BASE* principles. In BIM collaboration, a transaction should ultimately be considered as the information synchronization between parties involved in an exchange workflow. Schapke et al. [8] have discussed the advantages and downsides of optimistic and pessimistic concurrency control in BIM-based information exchange. According to their statements, implementing pessimistic concurrency control often requires locking mechanisms to prevent others from manipulating information in the scope of a running transaction. On the contrary, optimistic concurrency systems enable higher flexibility in decoupled, asynchronous settings. They conclude that optimistic concurrency appears suitable when the level of aggregation and interconnection across the managed information sets is low. We will discuss the aspect of aggregated information captured in BIM models and their subsequent exchange in interdisciplinary collaboration workflows in Section 5.

2.2. Approaches for data exchanges in BIM collaboration environments

Capturing and representing structured, highly interconnected design information of built facilities has been subject to extensive investigations over the past decade [14,15]. The application of affiliated techniques can range from routing problems in complex networks [16] to data fusion problems [17,18], and dependency representations in load-bearing systems [19]. In the scope of incremental version control, investigations related to object and product models and graph-based representations of such networks are of particular interest. For example, the transfer of procedural geometries for civil infrastructure assets has been extensively investigated by Vilgertshofer and Borrmann [20],Vilgertshofer [21]. For this problem, labeled, attributed, and directed graphs have been utilized to represent procedural geometry, which is then refined using graph transformation rules. Abualdenien and Borrmann [22] have picked up the idea of architectural patterns described in Postle [23]. They used graph transformations to capture design decisions and to apply them across different design models in a BIM authoring tool. Mattern and König [24] have also worked in a similar direction but focused on integrating design options into a graph representation reflecting the BIM model. Regarding design and variant communication, Zahedi et al. [25] presented an approach for decision capturing in early design stages using the BIM Collaboration Format (BCF). In essence, they employed ticketing systems to request and respond to specific properties to be present for a particular simulation or use case. For this, a feedback function was defined that transfers dedicated feature requests or suggestions on how a BIM model must be extended to fit a simulation. Even though their approach contributes to enhanced decision acceptance, it still lacks a proper (i.e., automated) integration into design tools. El-Diraby et al. [26] presented a framework that supports the management and reasoning about design decisions that have been made related to sustainability concerns in BIM-based design projects. Even though they do not consider changes in exchanged BIM models on the raw data level, their system supports identifying model components that gain high interest in interdisciplinary discussions by applying social-technical analysis techniques.

Esser et al. [10] have presented a thorough approach for transferring only the model updates instead of entire monolithic BIM files. Their core idea focuses on the representations of BIM models using labeled property graphs and the subsequent computation of update patches containing only the changes between two model versions. Also, Zhao et al. [27] have discussed a related approach. In contrast to the approach described in Esser et al. [10], they have used additional nodes to store each attribute of an entity instance, resulting in a greater number of nodes and edges than the One-to-One mapping of each class instance using node attributes. The general idea of reflecting IFC-based models in labeled property graphs has also been discussed by Zhu et al. [28]. Their concept focuses on the proper translation of a given serialized representation of a BIM model into a property graph. However, no further application of the achieved graph structure has been presented in their article.

Chuang and Yang [29] have recently presented an encompassing change identification framework that can determine deviations between two BIM models or a BIM model and laser scan data on a geometric level. Similar to the approaches presented in Collins et al. [30], they use graph-based scene representations for the geometries stored in a BIM model and a point cloud. In the analysis of geometric features, there are also other approaches that, in a broader sense, also use graph-based structures to compare models to the built reality [31].

Koch and Firmenich [32] researched versioning approaches for procedural building representations. Their approach is based on representing both design states and state transitions, thus enabling the exchange of change-oriented information through design steps denoted as modeling operations. Due to their procedural descriptions of the model alterations, ideas towards diverging model states have been tackled in their discourses.

A different approach of component-based synchronization is realized in the *Speckle* framework [33,34]. In contrast to other initiatives aiming to use well-defined data models, their system establishes direct peer-to-peer communication between various software tools widely established in the AEC domain. As a significant downside, however, all software products participating in such collaboration workflows must implement appropriate interfaces for sending and receiving update messages. Hence, their approach is at risk of becoming another data exchange standard and only partially solves the problems raised by collaborating in vendor-neutral BIM environments. Similar concerns can be raised by the approach Ruokamo and Rauno [35] have presented. Their concept features a Software Development Kit (SDK) for a jointly edited cloud-based BIM model that various applications can access. However, existing, internationally accepted, and widely adopted data models like the Industry Foundation Classes (IFC) are not adequately reflected in their system, which suggests a low acceptance of their approach.

Moving away from specific approaches that deal with time-evolving BIM models, Xue and Lu [36] have investigated the adoption of blockchain technologies in BIM collaboration systems. They combined blockchain techniques with interplanetary file system approaches and thus were able to capture model states securely. Xue and Lu [36] describe a three-layered framework capable of tracking changes between model stages and capturing these transactions in a blockchain. The actual increment calculation is carried out on a JSON-like data structure created based on IFC models, which can lead to a tremendous information loss about the actual object network.

2.3. Graph transformations

The concept of graph transformation (also denoted as graph rewriting) builds upon graph theory and introduces formal definitions to modify graphs. The general idea of graph rewriting is the mutation of a given host graph H into a resulting graph H' by applying transformation rules. The formulation of these rules builds upon morphism statements that express relationships between nodes and edges of different (sub-)graphs [37]. Detailed explanations and algebraic definitions have been published by Blomer et al. [38] and Ehrig et al. [39], which the reader is directed to for further information.

Several previous publications utilize graph transformation to express alteration processes that must be applied to a given network of objects [10,20,21]. However, these contributions have implicitly assumed a chronological and sequential order in which their transformation rules are applied. In turn, Eichhoff and Roller [40] have defined criteria to detect whether the result of two transformation rules, irrespective of the order in which they are applied to a host graph, is the same (illustrated in Fig. 3). Their work has essentially built upon the research published by Corradini [41].

As one of their key objectives, Eichhoff and Roller [40] investigated independence criteria, which led them to formulate morphism statements between two Double-Push-Out diagrams (i.e., if two transformations are mutually commutative). Transferring their conclusions to the problem area of diverging BIM model states, we can utilize two graph transformation rules that formally describe change applied to the BIM model. Accordingly, the vision of combining eventually diverging models closely aligns with their achievements. For graphs reflecting BIM models, however, more precise investigations are required to understand implications that may arise not only from a pure graph-based perspective, but also on the model and design intent level as depicted in Fig. 2. The latter will still require a comprehensive contribution of the users' expertise because of the extensive range of situations BIM models may contain.

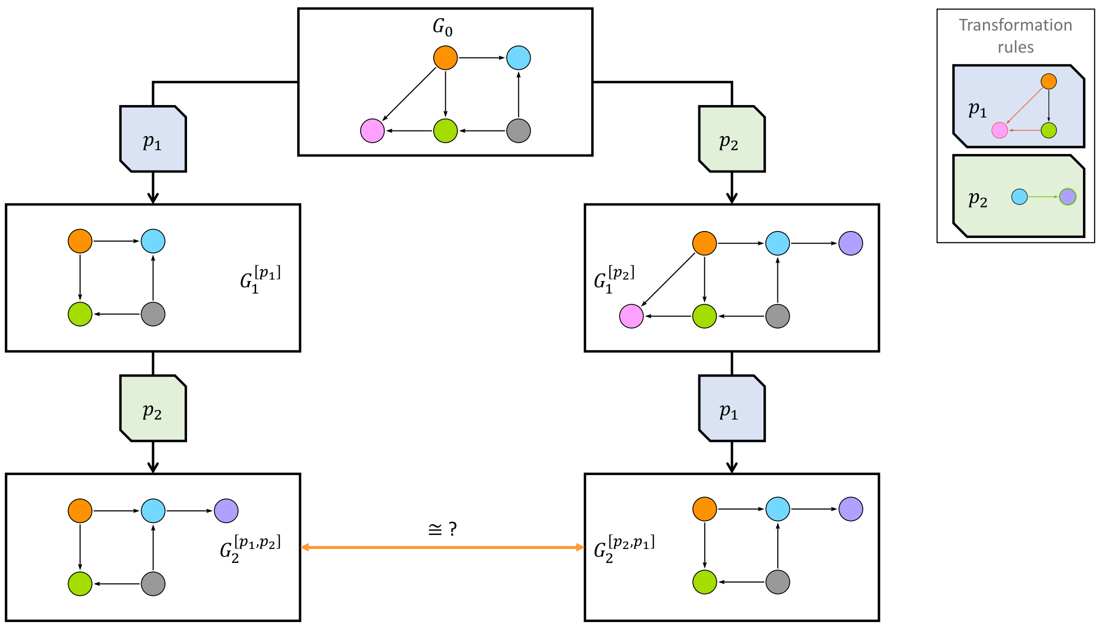


Fig. 3. Derivation confluence of graph transformation rules: Does it influence the resulting graph G_2, in which order the rules $p1$ and $p2$ were applied? (figure influenced by the work of Eichhoff and Roller [40]).

2.4. Identified research gap

The literature review has proven a great interest in graph-based knowledge representations to capture, exchange, and facilitate design decisions in building and civil infrastructure design. However, the existing approaches concerning graph transformations for incremental model updates only facilitate a very limited scope (e.g., only procedural geometry representations) or demand the execution of transformation rules in chronological order. None of the existing approaches has considered diverging model states and their potential recombination in a time-flexible manner. The fundamental works of Eichhoff and Roller [40] and Corradini [41] have unveiled formal transformation definitions and mutual dependencies across different graph transformation rules. Hence, the vision of combining non-chronological graph transformation rules with existing principles of model-based collaboration appears valid and is discussed further in the next chapters.

The following section summarizes preliminary work used as the foundation to motivate branching and merging concepts. The key objective is the development of confluence conditions under which a merge operation between diverging model states can be performed.

3. Merging diverging BIM model states using graph transformation techniques

3.1. Preliminary work

The research described in the following section builds upon crucial aspects already published in Esser et al. [10]. To ensure a complete understanding of the merging approach discussed in this paper, the following explanations recap the most relevant aspects concerning the representation of BIM models in labeled property graphs, the determination of incremental change between two model versions, and their exchange as incremental model patches using graph transformation techniques.

As discussed in Section 2, graph-based representations have been proven to be a suitable means for incremental updates applied to object networks. Graph theory can be used in various manifestations depending on the application domain and the technical systems. However, all graphs have in common that they are composed of a set of nodes and edges connecting the nodes. Edges may be directed or undirected and may potentially carry additional weights.

Labeled property graphs in particular follow this standard notation and enable the storage and interaction with directed, multi-labeled, attributed nodes and edges [42,43]. Accordingly, each node reflects an object, whereas edges are used to model relationships between two objects. All attributes specified for a dedicated object are attached to the node and filled according to the design information given by the BIM model. These capabilities align very well with object-oriented modeling principles, which provide the foundation for various data models currently used in the AEC domain. In essence, each class instance is reflected by a single node. All attributes specific to the instance in question are attached to the respective node as properties. Furthermore, relations between instances are modeled by directed edges in the graph. The name of the class the node is instantiating is stored in an additional attribute called *EntityType*. Additionally, the name of a relationship between two nodes or class instances, respectively, is captured in an attribute *relType*, which is attached to each edge in the graph. In labeled property graphs, properties must be in key–value pairs where the value must be of a simple data type (e.g., a Boolean, string, or numerical value). In addition to properties, labels can describe specific types of nodes and edges.

Fig. 4 illustrates the preliminary work of the authors that covers the representation of object networks stored inside a BIM model, their translation into labeled property graphs, the abstraction of changes between two model versions using the graph representations, and their formal transfer to a receiving unit including validation of successful patching [10].

An essential aspect of the framework is the independence from the data models to be managed in the version control system. This flexibility is accomplished by choosing a simple yet comprehensive graph meta-model. A graph meta-model defines characteristics such as allowed labels and attributes. In the case discussed here, a minimum set of characteristics was sought that could be applied to all data models to be considered in the BIM context. This analysis has led to three different nodes labels that are used to characterize nodes in the property graph:

- *PrimaryNodes*: These nodes carry a globally unique identifier as one of their properties that is assumed to be stable over all versions of a BIM model.

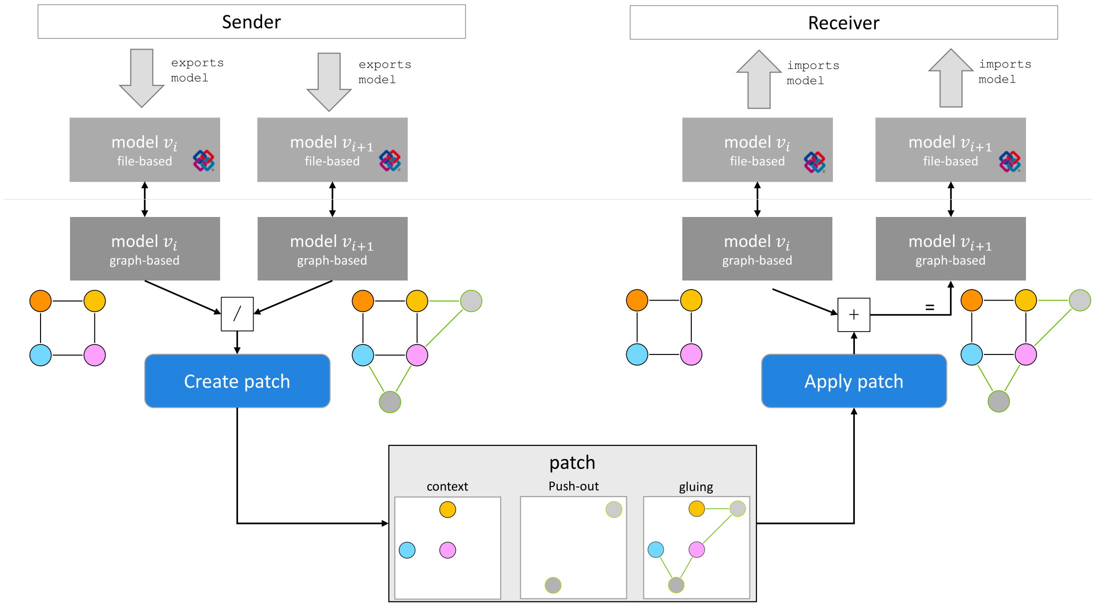


Fig. 4. The diff-and-patch workflow (influenced by Esser et al. [10]).

- *SecondaryNodes*: These nodes capture instances given in an object network that do not have a unique identifier. However, it must be satisfied that each secondary node is connected with at least one primary node (either directly or by a path).
- *ConnectionNodes*: These nodes are used to represent one-to-many or many-to-many relationships.

Class instances with unique identifiers establish a semantic skeleton in all of the investigated data schemata (including the Industry Foundation Classes (IFC), LandXML, CityGML, PlanPro, RailML, and others) to which several "resources" are associated. These resources are defined according to domain-specific requirements and may represent geometric shapes, material associations, costs, etc.

To calculate the incremental update between two versions of a BIM model, both files are translated into their graph representations and compared to identify the maximum common subgraph that is mutually isomorphic, reflecting the BIM model's unchanged parts. Subsequently, all other parts of the graphs that do not belong to the common subgraph must have been affected by modifications. Removed items turn up in subgraphs only present in the initial but not the updated graph. Accordingly, inserted objects are reflected in subgraphs that only exist in the updated graph representation but are not contained in the initial version. For both removed and inserted objects, suitable transformation rules are formulated that describe how the removal and insertion must be performed on an outdated graph. If model objects are present in both versions but have modified properties, these alterations are captured as semantic changes because they do not require changes in the topological shape of the graph. Removed and inserted subgraphs and property changes are then captured in a patch object δ.

To search for a specific structure in a graph, the notion of *patterns* and *pattern matching* is used. The term "pattern“ describes the assembly of nodes and edges forming a graph, whereas the term "matching“ relates to the actual search of the pattern inside a specific graph.

In the case of the diff-and-patch approach, three different patterns are captured and exchanged in a patch:

- The **context pattern** specifying parts in the host graph relevant to perform the insertion or removal operation without violating unaltered parts of the model (i.e., graph, respectively).
- The **push-out pattern** reflecting inserted and removed nodes and edges (i.e., representing added and deleted objects in the BIM model).
- The **gluing pattern** specifying how the push-out pattern is connected to the host graph.

Fig. 5 illustrates the application method of a patch, including all three patterns. First, the so-called context pattern must be matched in the graph that the update patch should be applied to. Then the push-out pattern is applied. All nodes and edges identified as removed are detached and deleted in this step. Furthermore, newly inserted nodes and edges are created. Finally, the newly inserted subgraphs are connected to the existing graph structure according to the edges specified in the gluing pattern. Because of the existence of secondary nodes that do not carry a stable, unique identifier, all edges that should be built from or towards these nodes must be expressed with patterns that at least contain one primary node. This way, it is ensured that the correct secondary node in question is unambiguously matched.

3.2. Types of pattern matching

There are various options to specify a pattern to be matched in a labeled property graph. In the scope of the approach presented in this article, the following two cases are relevant, which adopt the terms used in Gallagher [44]:

- Structural matching
- Semantic pattern matching

An illustrative example is depicted in Fig. 6. The graph G in question consists of four nodes marked with capital letters A, B, C, and D. Additionally, the edge set of G consists of four directed edges labeled with *1*, *2*, *3*, and *4*.

As the term implies, structural patterns focus on the topological structure of nodes and edges that should be detected in the graph. Later, this differentiation will be relevant to identifying the conditions that must be met to perform the merging operation (see Section 3.6).

Of course, combinations of structural and semantic pattern statements are possible. For example, the discussed example of the semantic pattern matching could be extended by semantic conditions for node z to match the semantic information $anotherAttribute$: "$anotherValue$". Naturally, such an enhanced pattern would result in only one matching: the graphlet $A \rightarrow B \rightarrow D$.

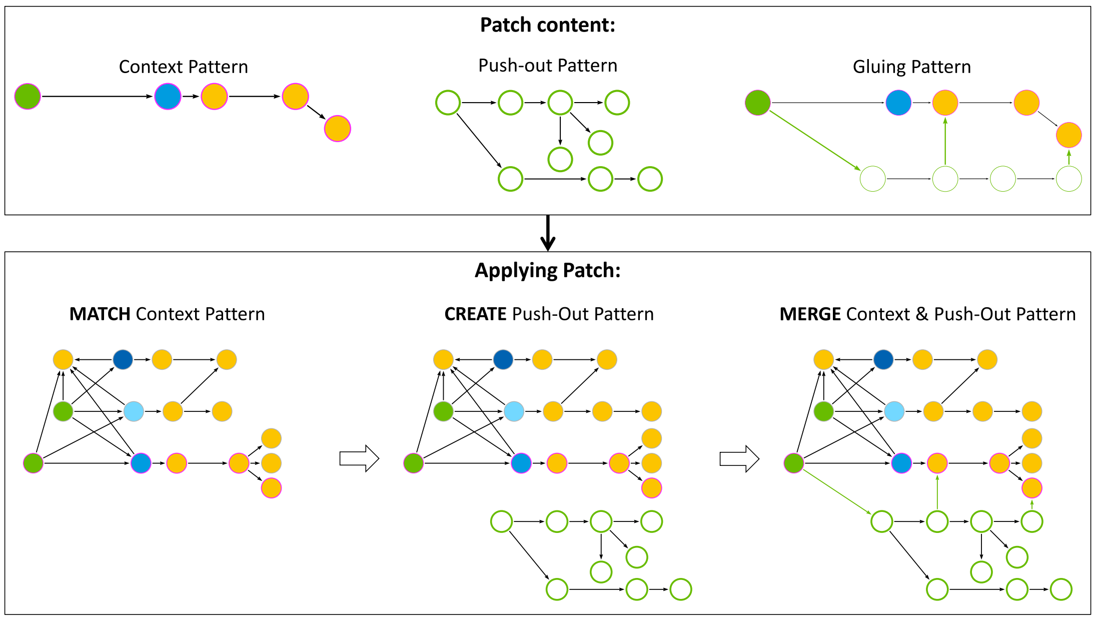
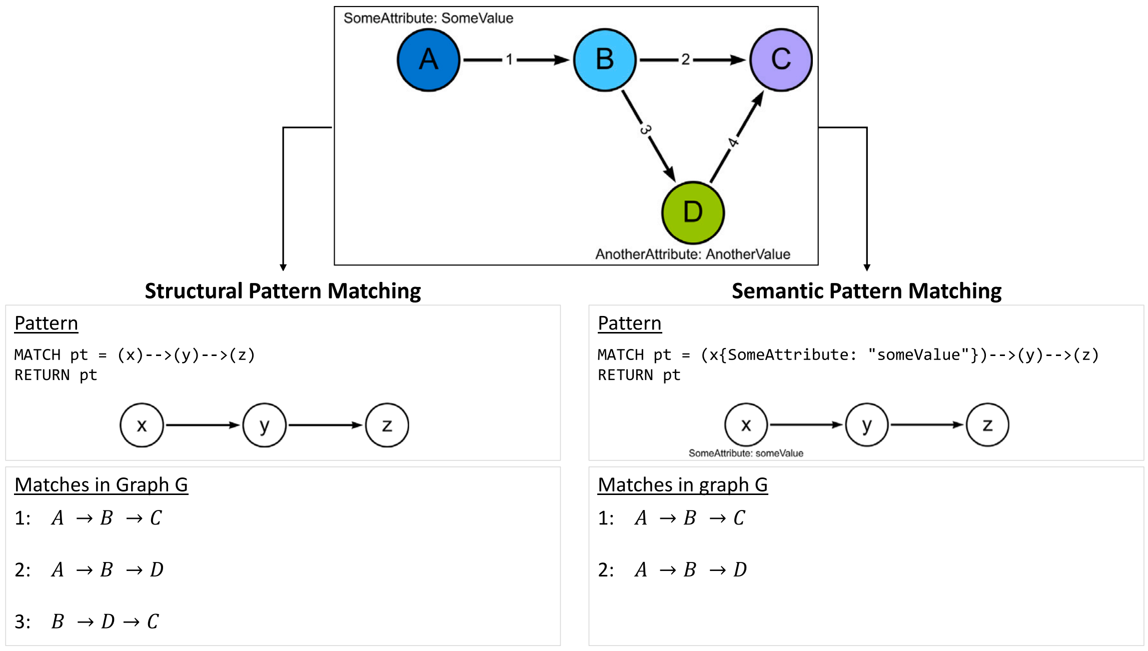


Fig. 5. Schematic illustration of the patch content and application procedure. Each patch comprises a context pattern, a push-out pattern, and a gluing pattern connecting the push-out part with the existing context in the host graph.



Structural Pattern Matching – Pattern:

```
MATCH pt = (x)-->(y)-->(z)
RETURN pt
```

Matches in Graph G:

1: $A \rightarrow B \rightarrow C$

2: $A \rightarrow B \rightarrow D$

3: $B \rightarrow D \rightarrow C$

Semantic Pattern Matching – Pattern:

```
MATCH pt = (x{SomeAttribute: "someValue"})-->(y)-->(z)
RETURN pt
```

Matches in graph G:

1: $A \rightarrow B \rightarrow C$

2: $A \rightarrow B \rightarrow D$

Fig. 6. Structural and semantic pattern matching. The pattern statements are expressed in the *Cypher* query language described in Francis et al. [43].

3.3. Extending diff-and-patch by checkout and merge

The version control of full-fledged BIM models presented in Esser et al. [10] is closely related to concepts implemented in the version control system Git [45]. As discussed in detail in Esser et al. [10], a pure text-based version control system cannot detect sensitive changes on the underlying object network that is exchanged by means of BIM models. A central issue exists in using identifiers that are not part of the actual domain but are added to represent associations between two instances in a textual manner. Furthermore, the engineering knowledge captured inside the textual representation can be serialized into instance models in a completely different order depending on the serialization strategy, resulting in seemingly diverging files representing the same design information. A text-based version control system would falsely recognize a completely changed model.

Nevertheless, on a more abstract level, Git provides good analogies for the approach presented here. Thus, we make use of its terminology where appropriate. Table 1 summarizes the commands and highlights deviations between both systems. The general vision of BIM-based

Table 1
Terminology comparison between Git [45] and the graph-based version control system described in [46].

Command	Git	Graph-based version control system
add	adds a file to the tracking or stages the file for an upcoming commit operation	translates a given (file-serialized) BIM model into its graph-based representation and stores it in the local graph storage
get	–	translates a graph-based representation of a BIM model back into a file-based representation (e.g., into a STEP P21, XML, or JSON encoding)
commit	records the changes made to the staged files and stores the snapshot as a commit	computes the differences between two graphs reflecting different versions of a BIM model and produces the patch
push	sends the commits locally produced and stored to another peer in the system (e.g., the remote server)	sends the patches to another peer (e.g., a server or another stakeholder in the project)
fetch	checks for updates on a remote peer	checks for updates on a remote peer
pull	pulls any commits of other peers into the local repository and applies the changes	pulls any patch objects from other peers and applies the transformations onto the graphs available in the local graph storage
checkout	switches to a specific branch	switches to a specific branch
merge	merges changes of another branch into the active (currently checked) branch	merges changes of another branch into the active (currently checked) branch

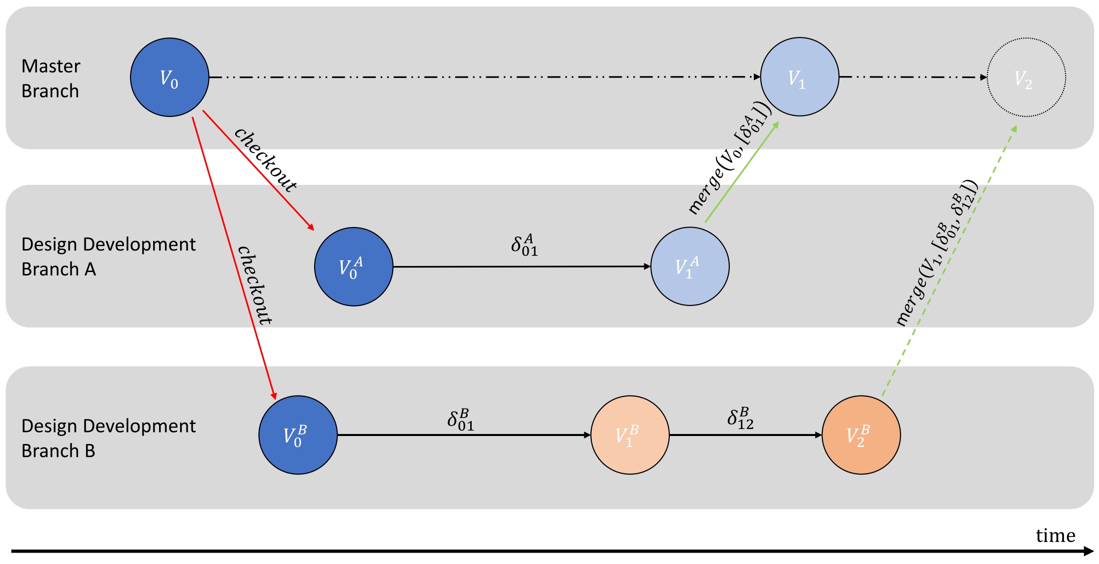


Fig. 7. Checkout and Merge (influenced by the work of Koch and Firmenich [32]).

version control and its role in upcoming BIM Level 3 collaboration has already been outlined in Esser et al. [46].

So far, it has been assumed that one author always issues incremental updates chronologically. This means an update patch can be consistently applied to the host (i.e., outdated) graph on the receiver's side. Only the model author is expected to issue new patches for a specific BIM model and its respective versions. In reality, however, this assumption does not reflect actual practices in multi-disciplinary engineering teams. Instead, it is often the case that different model variations are created to identify the best solution for a given engineering problem. In such cases, the user typically produces various files containing the BIM model with slight deviations. Once a decision is made, the final deliverable must be assembled, which is typically done in the authoring tool. A much better solution would be to enable the merge of different design variants without re-creating the final deliverable using the variant considerations. Hence, this behavior is closely aligned with a more optimistic concurrency control model. A user should be enabled to create and apply patches across potentially inconsistent (i.e., diverging) model stages.

To acknowledge this reality, we propose to extend the discussed diff-and-patch framework presented in Esser et al. [10] by two new concepts:

- to create *branches* and *checkout* an existing model state
- to merge *incremental* updates of one branch into another branch

Again, we make use of analogies to the text-based version control system Git, where both concepts exist and are well established.

3.4. Formal problem definition of merging diverged BIM model states

Fig. 7 expresses the envisioned branching and merging concept and extends the work of Eichhoff and Roller [40] by setting their approaches into the context of data models used in BIM workflows.

Given a BIM model exists in the state V_0, we state that a checkout operation copies the model state V_0 into a dedicated branch. A specific branch is specified using capital letters, resulting in A and B, respectively. Furthermore, the *checkout* operation does not change the graph specified in the source state, i.e., the resulting replicas are mutually isomorphic.

$$V_0 = V_0^A \tag{1}$$

and

$$V_0 = V_0^B \tag{2}$$

Now, modifications can be performed in each development branch independently, which leads to the formulation of patches δ_{ij}^A and δ_{ij}^B, respectively.

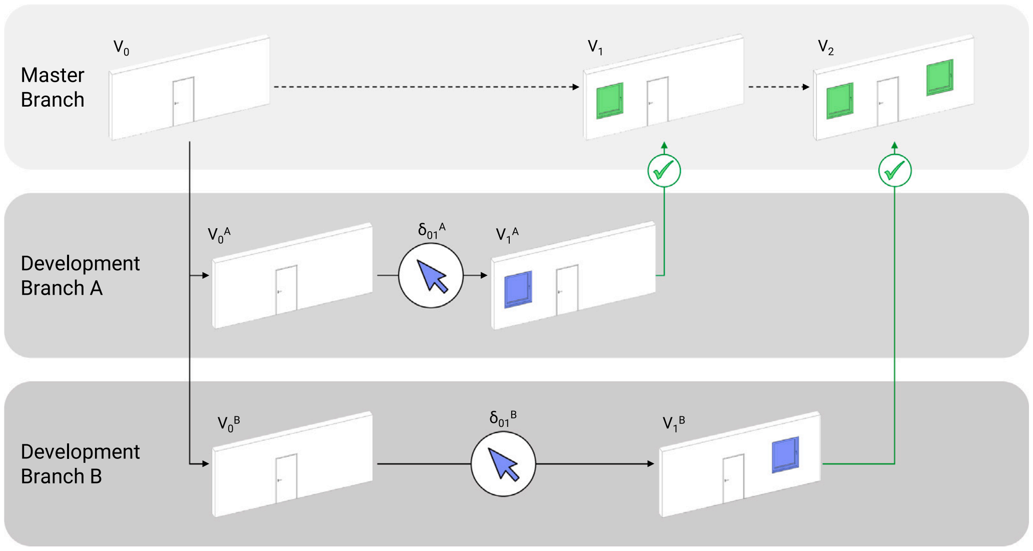


Fig. 8. Scenario 1: Two development branches, in which new windows are inserted into the BIM model.

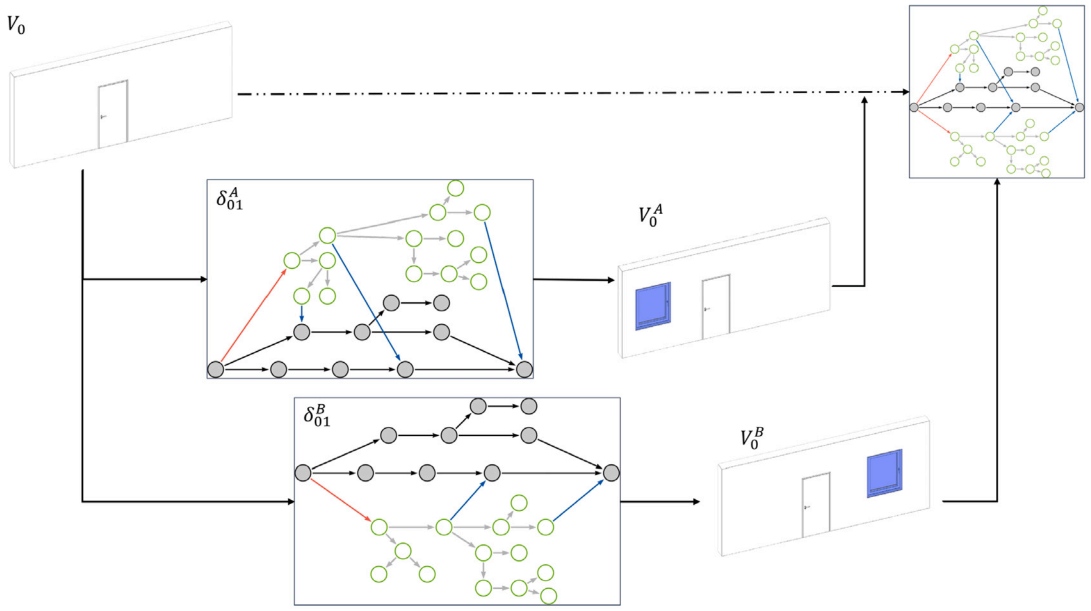


Fig. 9. Qualitative representation of the graph transformations encoded in δ_{01}^{A} and δ_{01}^{B} and the expected result in V_2. The gray nodes illustrate the context pattern required to perform both patch operations.

A patch δ contains a set of graph transformation rules that transfer the applied change.

We specify the merge operation to transfer a state from one branch to be transferred into another branch. A merge operation requires the following two inputs:

- A host state the merge should be applied onto.
- An ordered list of sequential patches $[\delta]$ that have been issued since the last checkout operation.

Fig. 7 depicts two merge operations. The first merge operation indicates that the changes made in development branch A are supposed to be merged into the master branch. As no changes have been applied to the master branch after the checkout into branch A, we can inevitably apply the patch δ_{01}^{A} on V_0 because of the given isomorphism between V_0 and V_0^A.

Despite the changes applied in branch A, two patches have been authored in branch B, resulting in the state V_2^B. Eventually, these updates shall be incorporated into the master branch as well. In contrast to the merge operation from branch A into the master, it is not ensured a priori if the merge operation from development branch B into the master branch can be performed immediately. The merge operation of patches issued in branch A into the master branch could have produced a state V_1 incompatible with the patches δ_{01}^{B} and δ_{12}^{B}.

The problem of merging diverging BIM model states can be formally expressed by the following problem statements:

1. Given the task of merging a list of patches into a model that has been altered since the initial checkout, how can we test the applicability of the patches?
2. How can we identify a merge conflict, and under which conditions can we perform a manual conflict resolution?

3.5. Possible checkout-merge scenarios

The following sections describe scenarios that may arise, resulting in diverging model states using separated, asynchronous development branches. The scenarios are explained using a simple BIM model containing a wall and a door in their initial state. In all cases, two branches called *Design Development Branch A* and *Design Development Branch B* are initialized. Using the notion of *checkout*, both branches feature the same model state as their initial state.

All investigations refer to the level of the raw object networks and do not necessarily reflect a meaningful design or a comprehensive engineering result. Hence, additional consistency checking concerning the design intents reflected in the graphs is required. We will discuss this aspect in more detail in Section 5.

3.5.1. Scenario 1: Automated merging

In scenario 1, we investigate the situation illustrated in Fig. 8. In *Development Branch A*, a new window has been added to the left of the door. In *Development Branch B*, on the other hand, a new window was inserted on the right side of the door. Both components require the wall as their hosting element.

Analyzing the object networks in detail, it can be identified that both insertion operations consist of a push-out part that represents the newly inserted window, its placement, geometric representation, the required voiding body, and associated semantic information. The context pattern specifies the wall and other model resources required to ensure the correct window insertion into the model. This may include aspects like the geometric context, the positioning reference of the new window, or the correct assignment of units. Accordingly, the gluing pattern is supposed to connect the subgraph reflecting the new window with all existing objects already stored in the graph to which the patch is applied.

Fig. 9 illustrates the patch application qualitatively. The nodes drawn in green show the newly inserted subgraph, whereas the red and green edges are the patch's gluing parts. As anticipated, both operations require specific nodes to exist in the host graph such that the inserted subgraphs can be connected to the existing graph successfully and unambiguously. However, the patches independently issued in branches A and B can be merged as they do not influence the context structure they rely on mutually. Hence, after merging both patches onto the initial graph $V_0 i$, the expected result is a model containing the wall, the door, and both windows.

3.5.2. Scenario 2: User decision required

In this scenario, two semantic changes (as opposed to merely structural changes, see Section 3.1) have been applied to the subgraph reflecting the door's position in the object network. In *Development Branch A*, the modeler has decided to move the door to the left, which leads to the patch δ_{01}^{A}. In contrast, the door has been moved to the right in *Development Branch B*, resulting in a corresponding patch δ_{01}^{B}. Hence, two diverging model states have been authored that are depicted in Fig. 10. We investigate if the patch δ_{01}^{B} issued in *Development Branch B* can be merged into the model state V_1, resulting from merging δ_{01}^{A} from *Development Branch A* previously.

In contrast to the first scenario, the applied changes lead to semantic modifications of nodes that are present in all nodes. Hence, neither nodes nor edges are inserted or removed, but the node attributes specifying the position of the door must be altered. Fig. 11 shows the context pattern that must be met to perform the patch δ_{01}^{B}. As described in Section 3.2, the node to be altered can be expressed by a semantic pattern, including the door position given in the initial model version V_0 and V_0^{B}, respectively. Even though the subgraph reflecting the position of the door is still structurally present in the graph, the semantic matching of the context pattern given in δ_{01}^{B} is going to fail when δ_{01}^{B} should be merged into the model version V_1 (i.e, the model state after the patch δ_{01}^{A} has been already merged).

A possible solution is to soften the pattern statement and to primarily focus on the structural aspect of the desired context pattern to be matched. Suppose the pattern query is reduced by any attributes specific to the model. In that case, a successful yet unique matching can identify the node that carries the position of the door. However, an automated overwrite of the value stored in the *Coordinates* attribute might not be intended. Therefore, we conclude that differentiating between the successful execution of exact and softened pattern matching provides a comprehensive means of diverging patches that can be merged without user interaction. The user must be asked to resolve conflicting changes.

3.5.3. Scenario 3: Failing merge operation

In a third scenario, the second scenario is modified such that the door is removed in *Development Branch A*. Accordingly, merging the patch δ_{01}^{A} into the master branch is still possible. However, when considering merging the position modification authored in *Development Branch B* and captured in the patch δ_{01}^{B} into the master branch, the operation must fail because the door is not available in the model state V_1 anymore. The problem is illustrated in Fig. 12.

3.6. Formalization and generalization of the merge problem

Given the scenarios mentioned above, a formal description is derived under which conditions a merge operation of diverging model states can be performed without conflicts. Furthermore, scenarios related to the illustrated scenario 2 described in Section 3.5.2 should be generalized such that a simple user decision can resolve identified conflicts. Fig. 13 provides an overview under which conditions (i) merging is possible and (ii) merge resolution is possible by a simple user decision. In all other cases, a more comprehensive analysis is required to identify the clashing changes encapsulated in different patches.

From these investigations, we conclude the following conditions for the merging problem:

- A conflict-free merging of patches is expected if all context patterns specified in the patches in question can be matched using strict semantic pattern statements.
- A user decision is required if the context patterns cannot be matched semantically but only structurally.
- Further investigation and communication with the patch authors are required in any other case.

As the amount of modifications per patch is not regimented, combinations of the mentioned situations can appear. In general, the chance for a conflict-free merge rises with a high frequency of information exchange and updates limited to rather small modifications. By contrast, long transactions typically result in a larger amount of conflicts to be resolved at merging time. Similar to the idea of schema-agnostic version control, the timing of patch issuing and integration cannot be advised in a general manner. Rather, the frequency of committed changes and the amount of change information inside each patch must be chosen wisely, depending on the specific circumstances.

Another aspect that requires extensive input by the system users is the maintenance of concurrent branches. Obviously, the fewer the branches that are opened in the system, the fewer the model variants that may exist and must be eventually merged again. According to the definition of the checkout operation, we always ensure that each version-controlled BIM model has a predecessor version. Hence, we can trace back the origin of model versions at any time and reproduce outdated model versions by applying the patches inversely.

If an entirely new BIM model is added to the version tracking, an initial transfer of its graph representation is required. In this case, the delta contains only insertion operations and performs a simple model copy. All operations described in Table 1 can be applied.

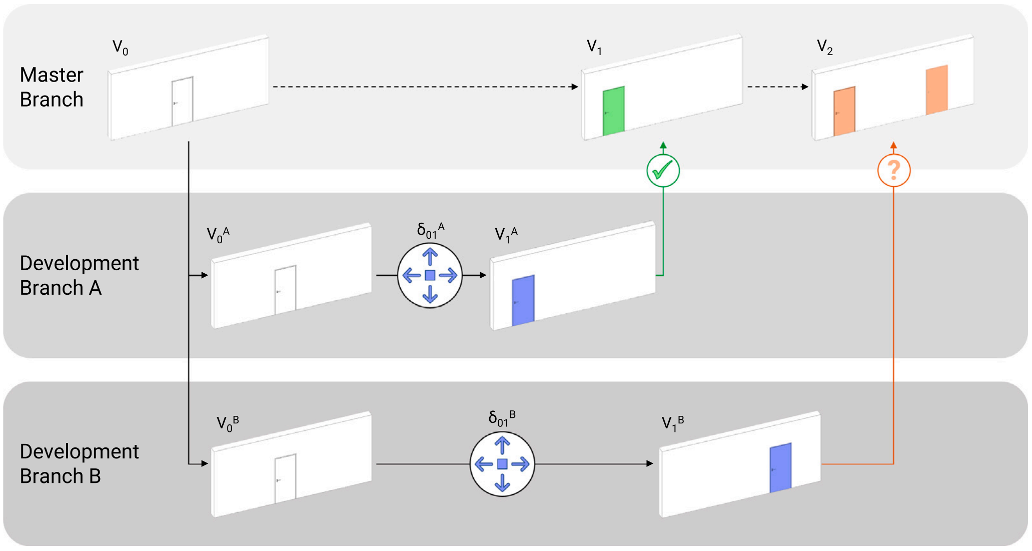
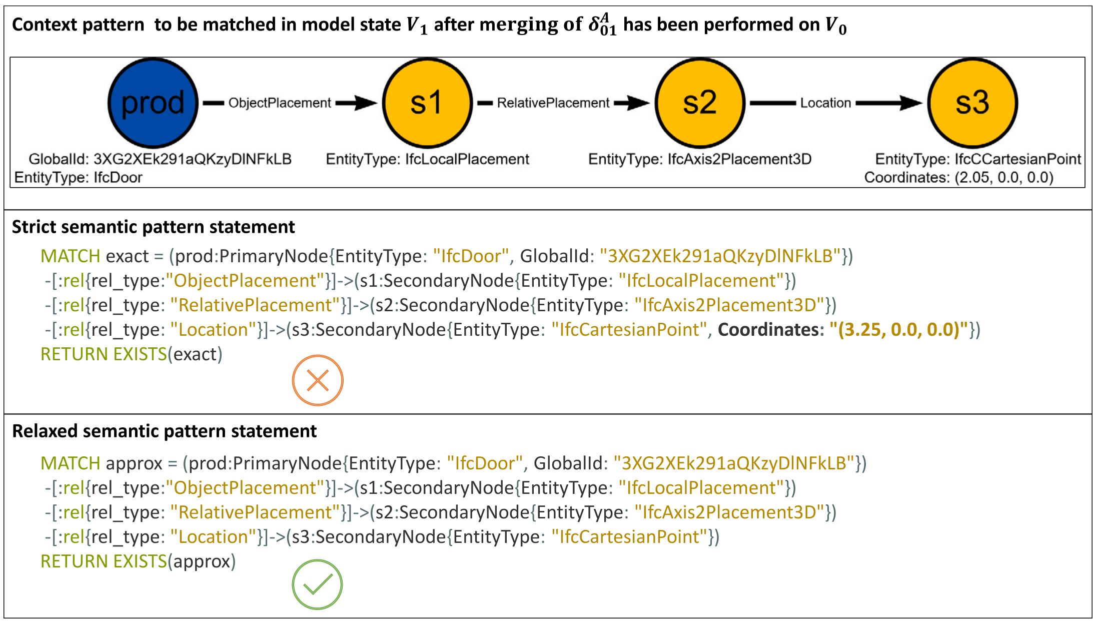


Fig. 10. Scenario 2: User interaction is required. Either the patch δ_{01}^B issued in *Development Branch B* is chosen to overwrite V_1 and thus withdraws the changes captured in patch δ_{01}^A or the user decides to stick to the model version V_1.


Context pattern to be matched in model state V_1 after merging of δ_{01}^A has been performed on V_0
prod
ObjectPlacement
s1
RelativePlacement
s2
Location
s3
GlobalId: 3XG2XEk291aQKzyDINFkLB
EntityType: IfcDoor
EntityType: IfcLocalPlacement
EntityType: IfcAxis2Placement3D
EntityType: IfcCCartesianPoint
Coordinates: (2.05, 0.0, 0.0)

Strict semantic pattern statement

```
MATCH exact = (prod:PrimaryNode{EntityType: "IfcDoor", GlobalId: "3XG2XEk291aQKzyDINFkLB"})
 -[:rel{rel_type:"ObjectPlacement"}]->(s1:SecondaryNode{EntityType: "IfcLocalPlacement"})
 -[:rel{rel_type: "RelativePlacement"}]->(s2:SecondaryNode{EntityType: "IfcAxis2Placement3D"})
 -[:rel{rel_type: "Location"}]->(s3:SecondaryNode{EntityType: "IfcCartesianPoint", Coordinates: "(3.25, 0.0, 0.0)"})
RETURN EXISTS(exact)
```

Relaxed semantic pattern statement

```
MATCH approx = (prod:PrimaryNode{EntityType: "IfcDoor", GlobalId: "3XG2XEk291aQKzyDINFkLB"})
 -[:rel{rel_type:"ObjectPlacement"}]->(s1:SecondaryNode{EntityType: "IfcLocalPlacement"})
 -[:rel{rel_type: "RelativePlacement"}]->(s2:SecondaryNode{EntityType: "IfcAxis2Placement3D"})
 -[:rel{rel_type: "Location"}]->(s3:SecondaryNode{EntityType: "IfcCartesianPoint"})
RETURN EXISTS(approx)
```

Fig. 11. Pattern matching: semantic-sensitive versus structural matching.

4. Cross-discipline collaboration

As the previous explanations primarily focus on the pure analysis of the object networks captured and exchanged using BIM models (see Fig. 2), this section aims to draw attention to possible application fields and discuss the correlation between the merging conditions and their relation to the BIM models and the respective design intent.

Following the motivation stated in Section 1, the opportunity to work asynchronously in decoupled design environments fits well with the mechanisms of how BIM-based design is currently carried out in large design teams. Using the formal notion of graph transformations as the technical backbone, formalizing model increments enables a schema-independent practice that can be applied to many data models employed in BIM-based data exchange workflows. The proposed methodology provides powerful means to allow each team member to work independently from the other team members. As long as the models to be updated (or, more precisely, their graph representations) provide the required context structure to perform the patching, our framework allows engineers and designers to combine different model variants into a joint deliverable. As the proposed system is closely aligned with the transaction principles of *BASE*, users of the system will potentially conceive the new flexibility as an enhancement but not as a significant change in current practices.

The application scenarios described in Sections 3.5.1 and 3.5.2 have focused on design interactions related to design collaboration within one discipline. However, the principles of checkout and merging can also support interdisciplinary collaboration among multiple project stakeholders and can revolutionize the setup of overall coordination models. As mentioned in the introduction, coordination models are formed by superimposing elements of several discipline models. However, no integration mechanism so far enables a tight combination of the same or similar elements.

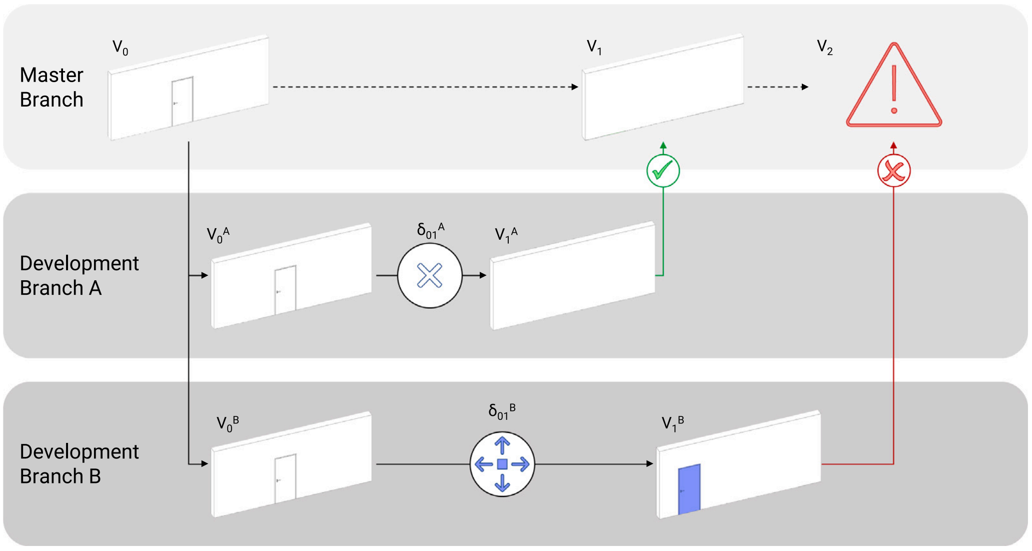


Fig. 12. Scenario 3: Failing Merge operation: The door has been removed in *Development Branch A*, and the changes have been merged, resulting in the model state V_1. Subsequently, the merge of the patches issued in *Development Branch B* fails because of the missing subgraph to which the modification should be applied.

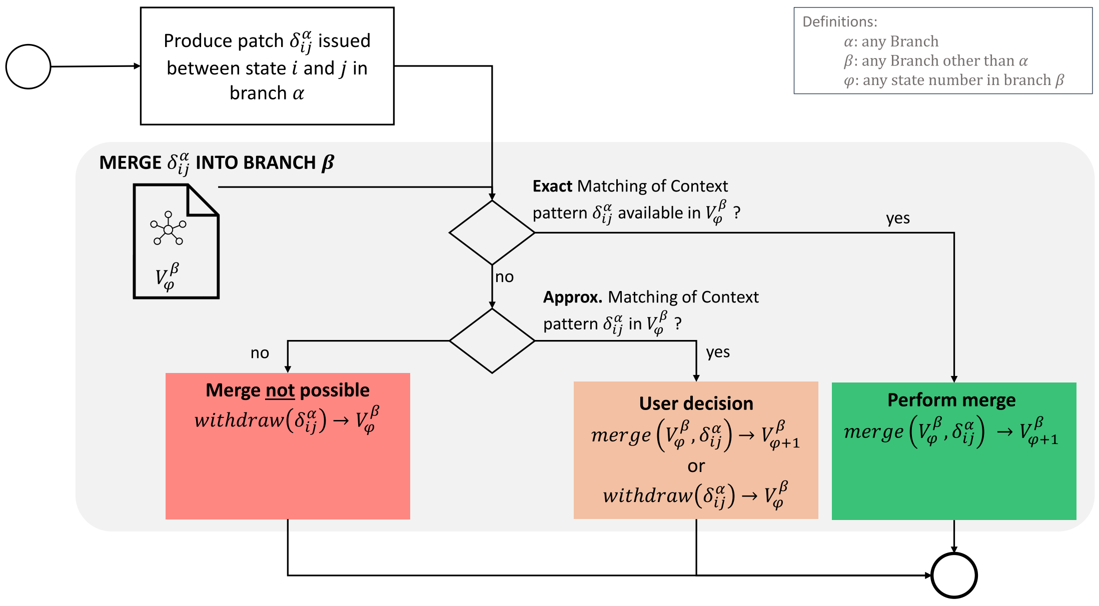

Produce patch δ_{ij}^{α} issued between state i and j in branch α
Definitions:
α: any Branch
β: any Branch other than α
φ: any state number in branch β
MERGE δ_{ij}^{α} INTO BRANCH β
V_{φ}^{β}
Exact Matching of Context pattern δ_{ij}^{α} available in V_{φ}^{β} ?
yes
no
Approx. Matching of Context pattern δ_{ij}^{α} in V_{φ}^{β} ?
no
yes
Merge not possible
$withdraw(\delta_{ij}^{\alpha}) \rightarrow V_{\varphi}^{\beta}$
User decision
$merge(V_{\varphi}^{\beta}, \delta_{ij}^{\alpha}) \rightarrow V_{\varphi+1}^{\beta}$
or
$withdraw(\delta_{ij}^{\alpha}) \rightarrow V_{\varphi}^{\beta}$
Perform merge
$merge(V_{\varphi}^{\beta}, \delta_{ij}^{\alpha}) \rightarrow V_{\varphi+1}^{\beta}$

Fig. 13. Formal description under which conditions a merge operation between diverging model states (i) can be performed without further investigations, (ii) require user interaction, or (iii) cannot be performed due to unsolvable context manipulations.

To overcome this limitation, Fig. 14 illustrates a potential system setup using the notions of branching and merging in a project-wide context. To ensure the ability to merge various discipline models that are authored and curated independently, it appears sufficient to initialize the overall collaboration model with a baseline model that already contains minimal project-wide information and definitions. Such a baseline model comprises aspects like unit settings, definitions related to geometric contexts, or any other data that any domain must consider. Each discipline subsequently creates branches of the baseline model and develops it into its discipline-specific domain model. In this process, each discipline team can perform version control of their model variants in their local, independent repository. Once a shareable state of a discipline model is reached, the authored model changes issued locally can eventually be merged into the branches maintained on any remote branch, e.g., the project-wide repository.

The envisioned system setup unveils several advantages. First and foremost, detailed tracking of deliverables and change management across an entire project becomes possible and enables direct reasoning on the actual changes contained in a specific patch. Furthermore, automated consistency checks can be performed every time a new contribution is about to be integrated into the coordination model. In case of conflicts, all parties that created competing patches can be notified directly to resolve conflicts. To resolve a conflict, additional patches may be created, which contain model updates that transform the coordination model into a sufficient state again. Hence, not only the conflict identification but also the agreed resolution is traced correctly in the system.

An example of the envisioned interdisciplinary collaboration system is presented in Fig. 15, which extends the introductory example provided in Fig. 1. The presented merging strategy assumes an *all-or-nothing* approach, meaning that a patch can be either applied to its full extent or it gets entirely withdrawn. Hence, to selectively choose and merge building components inserted in the two design variants, individual patches are issued for inserting the walls in design variant

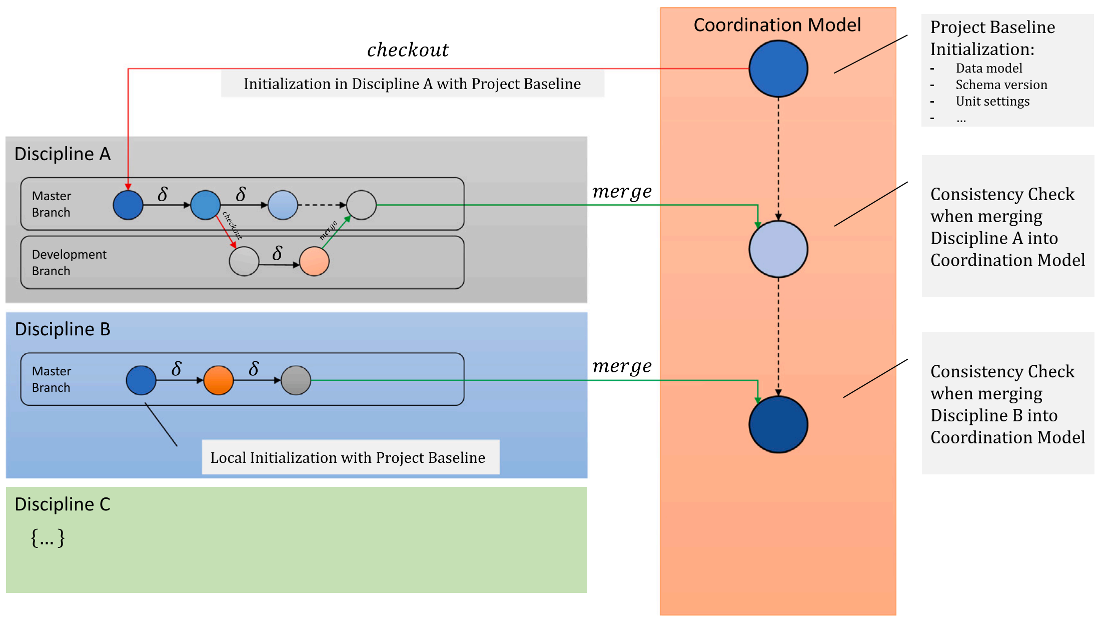

Coordination Model
checkout
Initialization in Discipline A with Project Baseline
Project Baseline Initialization:
- Data model
- Schema version
- Unit settings
- ...
Discipline A
Master Branch
Development Branch
δ
checkout
merge
merge
Consistency Check when merging Discipline A into Coordination Model
Discipline B
Master Branch
Local Initialization with Project Baseline
Consistency Check when merging Discipline B into Coordination Model
Discipline C
{...}

Fig. 14. Checkout and Merge in a global project context. Each discipline can perform local branching and merging. Additionally, the project-wide data exchange is performed as a merge operation into a central coordination model.

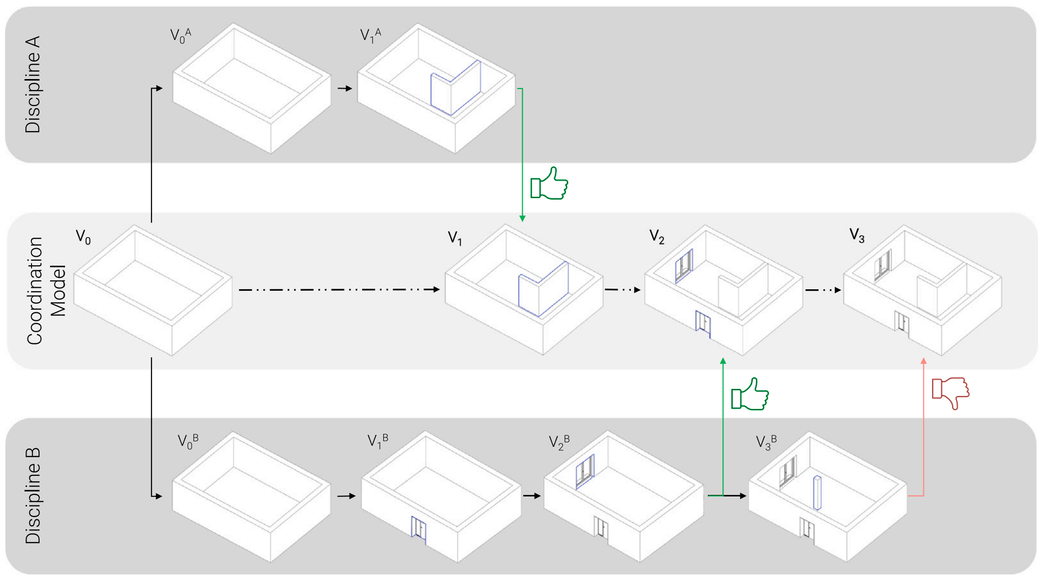


Fig. 15. Merging concurrent design variants into coordination models: A detailed view on the design fusion problem stated in Fig. 1. Green arrows indicate an accepted merge operation, whereas the user may decline the merge indicated by the red arrow due to the upcoming geometric conflict between the walls and the column.

A, and the windows, the door, and the column in design variant B, respectively. This is realized through various incremental versions in both disciplines. Of course, each discipline can additionally use branching and merge in its local scope (which has not been illustrated in Fig. 15).

In total, three merging operations are considered in the given scenario:

1. Merging model version 1 issued in discipline A into the coordination model (containing new inner walls)
2. Merging model version 2 issued in discipline B into the coordination model (containing the inserted door and window)
3. Merging model version 3 issued in discipline B into the coordination model (containing the inserted column)

As extensively discussed in Section 1, collaboration in distinct discipline models and superimposing them into a coordination model is still the better choice over joint editing of a central model. Accordingly, the responsibility to perform a merge operation cannot be fully automated but requires the understanding of the system operator. In the example illustrated in Fig. 15, the user might decide to merge just the first two cases and decline the merge of the column inserted in V_3^B because it will produce a geometric conflict with the walls already merged from discipline A. From a pure graph-based perspective, however, even the column insertion would result in a well-formed coordination model.

In the case that the geometric conflict between the wall and the column is not anticipated at merge time but gets unveiled in a subsequent model check, the proposed workflow would be to checkout V_2 of the coordination model and file another patch that either removes a component or moves one component such that the geometric clash is resolved. Alternatively, merged models can be checked automatically through clash detection routines provided by the central hub, declining a merge that does produce errors. This is in analogy to automated compiler runs provided through the "continuous integration" paradigm implemented by central code repositories.

In any case, design and coordination decisions are not anticipated to be fully automated. Rather, the diff-patch-merge system tempts to provide quicker access to model parts that require dedicated checking and care.

5. Discussion & limitations

The discussed concept unveils great potential to integrate diverging model variants into a joint deliverable. The presented diff-patch-merge approach contributes to the integration of partial models on the level of their underlying object networks. Even though the illustrated scenarios in Section 3.5 have demonstrated great potential, some deficiencies require further consideration.

5.1. Consistency checks on model and design intent level

As outlined, the merging as defined in Section 3.6 inserts or removes the specified subgraphs into or from the host graph without further analysis. The removal and insertion are performed once the context pattern has been successfully matched. However, the formal matching of the patterns in question only secures a correct execution on the level of the graph representations. In turn, successfully merging two branches does not necessarily lead to a meaningful yet correct representation at the higher abstraction layers depicted in Fig. 2. For example, there is no consistency check on the pure graph level that checks for potential geometric clashes or any other design issue. In scenario 1 discussed in Section 3.5.1, the transformations inside each patch could be applied successfully even if both windows overlap. To understand such clashes, the information represented by the graph must be interpreted, which is not seen as a core functionality of a classical version control system.

Drawing the analogy again to text-based version control systems, this issue is present there as well. Git, as the most popular version control system, offers comprehensive tools to stage, commit, and exchange increments of text files. However, Git does not provide any logic to resolve dependencies across different code files or syntactic checks of the version-controlled files, let alone the semantic correctness of the code they contain. In turn, additional software applications like Integrated Programming Environments (IDEs) are employed to support the developer in writing correct code. Moreover, modern collaboration platforms such as GitHub offer automated testing of committed code to ensure the correctness of any submitted code. We propose to transfer this concept to the presented graph-based version management accordingly. Various software vendors like Solibri [47] or ThinkProject [48], among others, already provide powerful software tools for the automated checking of BIM models. Hence, we foresee aligning the presented graph-based version control system operating on the pure graph-based representations with established principles of model checking to check the committed changes in BIM models against defined rules, ensuring consistency on the model view and design intent level.

5.2. Normalization across patches

In addition to the need for comprehensive consistency management, the conditions for merging operations illustrated in Fig. 13 assume that each patch produces specific, independent alterations. Due to the asynchronous and decoupled nature of branching, it may occasionally happen that two patches authored in separated branches are inserting similar graphs into the host graph. More precisely, the term "similarity" here refers to situations where all three patterns (pushout, glue, and context pattern) are homomorphic between two patches. In practical terms, situations may occur where two branches should be merged, which insert the same objects into the BIM model using similar resources (same geometric representation, same positioning, same material definitions, ...). There is a great risk of injecting redundant information into the overall model (and its graph representation, respectively) in these cases.

Fig. 16 illustrates the problem in a conceptual example. Two patches δ^A_{01} and δ^B_{01} are considered and merged together. In both settings, the patches have been issued with respect to the initial graph G. In the patch δ^A_{01}, the nodes D, E, and G have been inserted. In contrast, patch δ^B_{01} has issued the node insertion of D, E, and F. Specific attention must be now paid to the subgraph spanned by the nodes D and E and its directed, connecting edge. This subgraph is present in patches δ^A_{01} and δ^B_{01} and connects to the same node in the host graph G. Accordingly, merging both patches naturally produces the graphlet $D \rightarrow E$ to be present twice.

Although the duplicated insertion of similar subgraphs is not a severe problem on the structural level, two further facets must be considered. The duplicated insertion of subgraphs leads to a denormalization of the object network. The overall size of the graph increases without carrying more design information. However, a more crucial issue arises once another patch demands the parts of the duplicated subgraphs at a later point in time. Given that this patch requires the context pattern $B \rightarrow D \rightarrow E$, the result for this matching statement would result in two fitting subgraphs and thus introduces an ambiguity. The assumption that any secondary node can be uniquely identified by a path to one primary node no longer appears valid (see Section 3.1 for detailed definitions).

Further pre-processing of the patterns specified inside each patch is required to overcome this deficiency. In the situation illustrated in Fig. 16, the problem can be resolved if it had been checked whether and which subgraphs specified in the push-out pattern of δ^B_{01} had been already inserted during the merging of δ^A_{01}. Then, it would have become clear that the subgraph if $D \rightarrow E$ has already been given in the target graph and should not be inserted again. Subsequently, this part of the push-out pattern can be moved into the context pattern of δ^B_{01} to ensure that the newly inserted node F is connected correctly to the rest of the graph.

The problem described can occur in various modeling situations where potentially only a subset of a push-out pattern has already been inserted by another previous patch or merge operation. In other cases, however, the duplicated insertion of repetitive subgraphs (and the subsequent denormalization of the graph) might be of explicit intent. Hence, it appears impossible to resolve this deficiency in a formal, generic manner. Instead, further consistency checks and reasoning of the design information actually encapsulated in a specific patch are required.

5.3. Partial application of patches while merging

Another limitation is the constraint of applying patches in an *all-or-nothing* fashion. Hence, the user must consider this aspect when defining a suitable commit strategy to separate the design decisions into suitable patches.

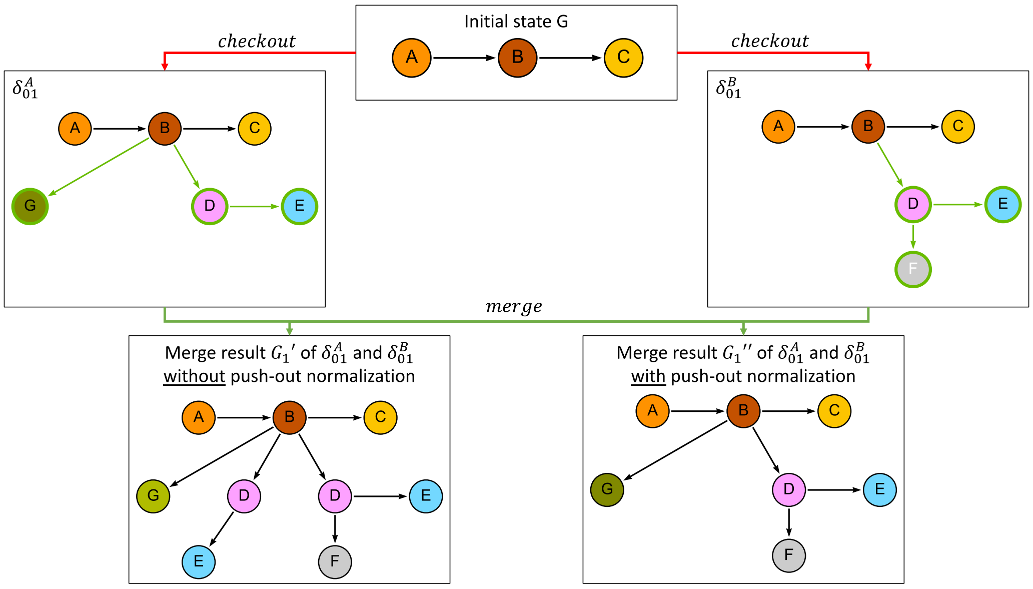

Initial state G
checkout
checkout
δ_{01}^A
δ_{01}^B
A
B
C
G
D
E
F
merge
Merge result G_1' of δ_{01}^A and δ_{01}^B without push-out normalization
Merge result G_1'' of δ_{01}^A and δ_{01}^B with push-out normalization

Fig. 16. Schematic illustration of push-out normalization problem. Given the two patches δ_{01}^A and δ_{01}^B, both insert a subgraph with the nodes D and E. According to the fashion mentioned above, the insertion would be carried out twice, resulting in the result graph G_1'. If the push-out patterns are considered in further pre-processing, the insertion of repetitive subgraphs can be prevented, which is illustrated in the graph plot of G_1''.

This limitation is vividly discussed in the example in Fig. 15. In the branch of discipline B, each insertion operation has been captured in a separate delta. Hence, the merge illustrated after version V_2^B requires the application of the patches δ_{01}^B, δ_{12}^B onto version V_1 of the coordination model. Also, the inserting operation of the column between versions V_2^B and V_3^B has been issued into a separate patch. Subsequently, the user can actively decide if merging after version V_3^B is intended or should not be performed.

In the case of the changes made by discipline A, only one patch exists that contains the insert operation of both inner walls. Therefore, either both walls can be accepted during the merge, or the entire merge operation must be declined. If only one wall were accepted, the proposed resolution would be to create another patch in the branch of discipline A, removing the wall that is not wanted. After issuing this removal, the proper merging of all patches available in discipline A can be performed again, resulting in only one of the two walls being inserted.

6. Conclusions

This paper has presented a comprehensive method that evaluates under which conditions the merging of diverging BIM model states is possible. For this, BIM models are represented using the notion of Labeled Property Graphs, which feature attributed labeled nodes and directed edges. Incremental version control is achieved using a diff-and-patch approach, capturing and exchanging only the delta between two model versions. Previous publications have assumed that model updates and the corresponding patches are always captured in sequential order. However, this assumption does not correctly fit reality. Instead, multiple diverging model states might be produced concurrently throughout a design process that should be eventually merged into a consistent joint coordination model.

To address this gap, the article has introduced a branch-and-merge concept that extends the incremental diff-and-patch approach of previous publications. Furthermore, a set of conditions has been derived from three representative scenarios that have motivated the merging problem on the instance level of graphs. This way, it can be now precisely determined if diverging states of a BIM model (i) can be merged without any conflict, (ii) require user decisions about which state should be used for further design tasks, or (iii) update patches influence each other in a way that automatic resolution is not possible. Thanks to the investigations made on a pure object network level, the approach is generic and can be applied to any data model following object-oriented concepts.

The workflow presented has been tested on small and large-scale models. In all cases, the core principles discussed in the paper remain the same for any model size and number of collaborators. Especially for projects with complex and large models and multiple disciplines involved, dedicated emphasis must be spent on sufficient synchronization frequencies, which must be chosen wisely according to the project circumstances. Regarding project management aspects, the eventual recombination appears particularly useful for project coordinators and managers. By merging updates authored in discipline models into a coordination model, conflicts between the individual domain contributions can be identified with much less effort and manual re-evaluation. Additionally, engineers and designers who must create many versions searching for optimal design solutions will greatly benefit from the enhanced tooling as branching and merging become available in their local workspaces. Hence, they can produce many *work-in-progress* models and then combine the most suitable variants to be shared with the project team. *Cherry-picking* of design decisions modeled in different versions becomes possible if the variant changes are committed into individual patches. This new technical means is not limited to a single software tool but could even be applied if different tools are in use that support vendor-open exchange standards.

Subsequent processing of the exchanged update patches can help gain additional information for collaborative workflows. Some update patches may have a higher impact and require more sophisticated treatment if they influence a multitude of contributors. In contrast, the project team can accept other updates without further discussion or reasoning.

The presented diff-patch-merge method is advantageous for various stakeholders in their daily organization and collaboration tasks. As the application of incremental update patches enables direct access to model changes applied, there is no need to compare different versions anymore manually. Having this knowledge available immediately, further automation of the information encapsulated in each patch can be established. For example, each participant can define specific filters that notify about changes made in foreign disciplines that may affect his or her design objectives. For incoming repetitive update patches, users will be able to define automation rules that directly propose sufficient model changes in their models. Especially for modifications of objects providing a particular service to elements modeled in other disciplines (e.g., hostings, voidings, and connectors), a direct link between these dependent objects can be established. In case of modifications, automation can be invoked to evaluate the impact caused by the incoming patch or propose a sufficient update of all involved models. In turn, patches with less relevance or no interdisciplinary dependency may be integrated without further investigation if no conflict is detected on the pure data level or any discipline-specific view. Indeed, creating such interdisciplinary links and correctly interpreting incoming updates and their impact on other domains exceeds the proposed system and requires future research work.

Besides enhancements to the clients' environments, integrating patches into the central project platforms can be equipped with additional quality gates before accepting incoming changes. Such checks can include geometric checks or predefined rules of semantic information to be present. Even the fulfillment of issues and change actions (e.g., captured in BCF files) appears to be further automated as the information of applied modification is now directly accessible. This way, an approval workflow similar to Git's *pull request* can be added to the overall collaboration ensuring further information quality. Again, further research activity is relevant to understand new challenges.

In summary, our system further enhances the overall collaboration and communication in the AEC industry by implementing improved change and update dissemination. This way, potential conflicts between involved experts can be detected much earlier and with less effort. Furthermore, possible solutions can be automatically proposed in the future, and earlier indications of necessary interactions can be provided.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We gratefully acknowledge the financial support of the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) in the frame of the programs FOR 2363 "Early BIM" and TRR 277 "Additive Manufacturing in Construction – The Challenge of Large Scale" (project no. 414265976). In addition, we would like to thank Autodesk, Inc., United States for its 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. 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, http://dx.doi.org/10.1007/978-3-319-92862-3_1.
[3] M. Oh, J. Lee, S.W. Hong, Y. Jeong, Integrated system for BIM-based collaborative design, Autom. Constr. 58 (2015) 196–206, http://dx.doi.org/10.1016/j.autcon.2015.07.015.
[4] Š. Jaud, S. Esser, S. Muhič, A. Borrmann, Development of IFC schema for infrastructure, in: Proceedings of the 6th International Conference SiBIM: Structured Data are New Gold, 2020, pp. 27–35, URL https://publications.cms.bgu.tum.de/2020_Jaud_siBIM.pdf, (last access: 2023-06-15).
[5] A. Borrmann, S. Muhič, J. Hyvärinen, T. Chipman, Š. Jaud, C. Castaing, C. Dumoulin, T. Liebich, L. Mol, The IFC-Bridge project – Extending the IFC standard to enable high-quality exchange of bridge information models, in: 2019 European Conference on Computing in Construction, 2019, pp. 377–386, http://dx.doi.org/10.35490/EC3.2019.193.
[6] British Standards Institution, PAS 1192-2: Specification for information management for the capital/delivery phase of construction projects using building information modelling, 2013, URL https://knowledge.bsigroup.com/products/specification-for-information-management-for-the-capital-delivery-phase-of-construction-projects-using-building-information-modelling-1/standard, (last access: 2023-07-25).
[7] 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, 2018, URL https://www.iso.org/standard/68078.html, (last access: 2023-07-25).
[8] S.-E. Schapke, J. Beetz, M. König, C. Koch, A. Borrmann, Collaborative data management, in: Building Information Modeling, Springer International Publishing, Cham, 2018, pp. 251–277, http://dx.doi.org/10.1007/978-3-319-92862-3_14.
[9] 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, http://dx.doi.org/10.1007/978-3-319-92862-3_15.
[10] S. Esser, S. Vilgertshofer, A. Borrmann, Graph-based version control for asynchronous BIM collaboration, Adv. Eng. Inform. 53 (2022) 101664, http://dx.doi.org/10.1016/j.aei.2022.101664.
[11] S.W. Sadiq, M.E. Orlowska, W. Sadiq, Specification and validation of process constraints for flexible workflows, Inf. Syst. 30 (2005) 349–378, http://dx.doi.org/10.1016/j.is.2004.05.002.
[12] J. Gray, The transaction concept: Virtues and limitations, in: Proceedings of Seventh International Conference on Very Large Databases, Tandem Computers Incorporated, 1981, pp. 144–154, URL http://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F21/handouts/papers/theTransactionConcept.pdf, (last access 2023-06-15).
[13] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2015, http://dx.doi.org/10.1016/b978-0-12-407192-6.00003-0.
[14] L. Kolbeck, S. Vilgertshofer, J. Abualdenien, A. Borrmann, Graph rewriting techniques in engineering design, Front. Built Environ. 7 (2022) 1–19, http://dx.doi.org/10.3389/fbuil.2021.815153.
[15] C. Voss, F. Petzold, S. Rudolph, Graph transformation in engineering design: an overview of the last decade, Artif. Intell. Eng. Des. Anal. Manuf. 37 (2023) e5, http://dx.doi.org/10.1017/S089006042200018X.
[16] A. Kneidl, A. Borrmann, D. Hartmann, Generation and use of sparse navigation graphs for microscopic pedestrian simulation models, Adv. Eng. Inform. 26 (2012) 669–680, http://dx.doi.org/10.1016/j.aei.2012.03.006.
[17] J. Hao, L. Zhao, J. Milisavljevic-Syed, Z. Ming, Integrating and navigating engineering design decision-related knowledge using decision knowledge graph, Adv. Eng. Inform. 50 (2021) 101366, http://dx.doi.org/10.1016/j.aei.2021.101366.
[18] J. Johansson, M. Contero, P. Company, F. Elgh, Supporting connectivism in knowledge based engineering with graph theory, filtering techniques and model quality assurance, Adv. Eng. Inform. 38 (2018) 252–263, http://dx.doi.org/10.1016/j.aei.2018.07.005.
[19] A. Justo, D. Lamas, A. Sánchez-Rodríguez, M. Soilán, B. Riveiro, Generating IFC-compliant models and structural graphs of truss bridges from dense point clouds, Autom. Constr. 149 (2023) 104786, http://dx.doi.org/10.1016/j.autcon.2023.104786.
[20] S. Vilgertshofer, A. Borrmann, Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models, Adv. Eng. Inform. 33 (2017) 502–515, http://dx.doi.org/10.1016/j.aei.2017.07.003.
[21] S. Vilgertshofer, Kopplung von Graphersetzung und parametrischer Modellierung zur Unterstützung des modellbasierten Entwerfens und der Erstellung mehrskaliger Modelle (Ph.D. thesis), Technical University of Munich, School of Engineering and Design, Munich, 2022, URL https://mediatum.ub.tum.de/doc/1687268, (last access: 2023-06-15).
[22] J. Abualdenien, A. Borrmann, PBG: A parametric building graph capturing and transferring detailing patterns of building models, in: Proc. of the CIB W78 Conference 2021, Luxembourg, 2021, pp. 11–15, URL https://itc.scix.net/pdfs/w78-2021-paper-001.pdf, (last access: 2023-06-15).
[23] B. Postle, On pattern languages, design patterns and evolution, New Des. Ideas 3 (2019) 44–52, URL http://jomardpublishing.com/UploadFiles/Files/journals/NDI/V3N1/PostleB.pdf, (last access 2023-06-15).
[24] H. Mattern, M. König, BIM-based modeling and management of design options at early planning phases, Adv. Eng. Inform. 38 (2018) 316–329, http://dx.doi.org/10.1016/j.aei.2018.08.007.
[25] A. Zahedi, J. Abualdenien, F. Petzold, A. Borrmann, Minimized communication protocol based on a multi-LOD meta-model for adaptive detailing of BIM models, in: P. Geyer, K. Allacker, M. Schevenels, F.D. Troyer, P. Pauwels (Eds.), Proc. of the 26th International Workshop on Intelligent Computing in Engineering 2019, Leuven, Belgium, 2019, pp. 1–10, URL https://ceur-ws.org/Vol-2394/paper06.pdf, (last access: 2023-06-15).

[26] T. El-Diraby, T. Krijnen, M. Papagelis, BIM-based collaborative design and socio-technical analytics of green buildings, Autom. Constr. 82 (2017) 59–74, http://dx.doi.org/10.1016/j.autcon.2017.06.004.

[27] Q. Zhao, Y. Li, X. Hei, M. Yang, A graph-based method for IFC data merging, Adv. Civ. Eng. 2020 (2020) 1–15, http://dx.doi.org/10.1155/2020/8782740.

[28] J. Zhu, P. Wu, X. Lei, IFC-graph for facilitating building information access and query, Autom. Constr. 148 (2023) 104778, http://dx.doi.org/10.1016/j.autcon.2023.104778.

[29] T.-Y. Chuang, M.-J. Yang, Change component identification of BIM models for facility management based on time-variant BIMs or point clouds, Autom. Constr. 147 (2023) 104731, http://dx.doi.org/10.1016/j.autcon.2022.104731.

[30] F.C. Collins, M. Ringsquandl, A. Braun, D.M. Hall, A. Borrmann, Shape encoding for semantic healing of design models and knowledge transfer to scan-to-BIM, Proc. Inst. Civ. Eng. 175 (2022) 160–180, http://dx.doi.org/10.1680/jsmic.21.00032.

[31] T. Meyer, A. Brunn, U. Stilla, Change detection for indoor construction progress monitoring based on BIM, point clouds and uncertainties, Autom. Constr. 141 (2022) 104442, http://dx.doi.org/10.1016/j.autcon.2022.104442.

[32] C. Koch, B. Firmenich, An approach to distributed building modeling on the basis of versions and changes, Adv. Eng. Inform. 25 (2011) 297–310, http://dx.doi.org/10.1016/j.aei.2010.12.001.

[33] P. Poinet, Enhancing Collaborative Practices in Architecture, Engineering and Construction through Multi-Scalar Modelling Methodologies (Ph.D. thesis), The Royal Danish Academy of Fine Arts, Schools of Architecture, Design and Conservation, 2019, URL https://adk.elsevierpure.com/en/publications/enhancing-collaborative-practices-in-architecture-engineering-and, (last access: 2023-06-15).

[34] P. Poinet, D. Stefanescu, E. Papadonikolaki, Collaborative workflows and version control through open-source and distributed common data environment, in: Lecture Notes in Civil Engineering, Vol. 98, Springer International Publishing, 2020, pp. 228–247, http://dx.doi.org/10.1007/978-3-030-51295-8_18.

[35] S. Ruokamo, H. Rauno, Single shared model approach for building information modelling, in: 37th International Symposium on Automation and Robotics in Construction (ISARC2020), Kitakyshu, Japan / online, 2020, pp. 240–247, http://dx.doi.org/10.22260/ISARC2020/0035.

[36] F. Xue, W. Lu, A semantic differential transaction approach to minimizing information redundancy for BIM and blockchain integration, Autom. Constr. 118 (2020) 103270, http://dx.doi.org/10.1016/j.autcon.2020.103270.

[37] G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1997, http://dx.doi.org/10.1142/3303.

[38] J. Blomer, R. Geiß, E. Jakumeit, The GrGen.NET user manual, 2013, URL http://www.info.uni-karlsruhe.de/software/grgen/GrGenNET-Manual.pdf, (last access 2023-06-15).

[39] H. Ehrig, U. Prange, G. Taentzer, Fundamental theory for typed attributed graph transformation, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 3256, (April) 2004, pp. 161–177, http://dx.doi.org/10.1007/978-3-540-30203-2_13.

[40] J.R. Eichhoff, D. Roller, Designing the same, but in different ways: Determinism in graph-rewriting systems for function-based design synthesis, J. Comput. Inf. Sci. Eng. 16 (2016) http://dx.doi.org/10.1115/1.4032576.

[41] A. Corradini, Concurrent graph and term graph rewriting, in: CONCUR '96: Concurrency Theory. CONCUR 1996. Lecture Notes in Computer Science, Vol. 1119, 1996, pp. 438–464, http://dx.doi.org/10.1007/3-540-61604-7_69.

[42] J. Hidders, A Graph-based Update Language for Object-Oriented Data Models (Ph.D. thesis), (2001) Eindhoven University of Technology, 2001, http://dx.doi.org/10.6100/IR551259.

[43] 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, New York, NY, USA, 2018, pp. 1433–1445, http://dx.doi.org/10.1145/3183713.3190657.

[44] B. Gallagher, Matching structure and semantics: A survey on graph-based pattern matching, in: AAAI Fall Symposium: Capturing and using Patterns for Evidence Detection (Vol. 45), 2006, pp. 43–53, URL https://cdn.aaai.org/Symposia/Fall/2006/FS-06-02/FS06-02-007.pdf, (last access 2023-06-15).

[45] S. Chacon, Pro Git, A Press, Berkeley, CA, 2009, http://dx.doi.org/10.1007/978-1-4302-1834-0.

[46] S. Esser, S. Vilgertshofer, A. Borrmann, A reference framework enabling temporal scalability of object-based synchronization in BIM level 3 systems, in: Proceedings of the 2023 European Conference on Computing in Construction and 40th International CIB W78 Conference, Heraklion, Greece, 2023, http://dx.doi.org/10.35490/EC3.2023.177.

[47] Solibri, Solibri office, 2023, URL https://www.solibri.com, (visited on 2023-02-17).

[48] Thinkproject, DESITE BIM, 2023, URL https://thinkproject.com/de/produkte/desite-bim/, (visited on 2023-02-17).