2026 European Conference on Computing in Construction
Corfu, Greece
July 12–15, 2026

ENRICHING VERSION INCREMENTS WITH ENGINEERING INTENT FOR ENHANCED BIM MODEL CHECKING

Sebastian Esser [1,2], Jiabin Wu [1,2], Suhyung Jang [1,2], Lena Fahrner [3], Robert Amor [4], and André Borrmann [1,2]
[1] Chair of Computing in Civil and Building Engineering, Technical University of Munich, Germany
[2] TUM Georg Nemetschek Institute, Technical University of Munich, Germany
[3] Landmarken AG, Aachen, Germany
[4] University of Auckland, School of Computer Science, Auckland, New Zealand

Abstract

BIM-based model checking helps improve the quality of design data throughout the planning phase of a built asset. However, rule-based checking can be time-consuming for large, frequently updated BIM models. Increment descriptors for BIM models are enriched with engineering intent, thereby unveiling additional information about the changes made. The proposed method enhances model-checking efficiency by executing only the rules affected by changes, rather than reprocessing the entire model. A case study demonstrates that selective rule triggering reduces redundant checks by re-executing only the rules affected by a given change, automatically identifying which open issues a model update resolves, thereby improving traceability between design edits and compliance outcomes.

Introduction

In recent years, substantial progress has been achieved in coordinating interdisciplinary models in the Architecture, Engineering, and Construction (AEC) industry, largely through the adoption of Building Information Modeling (BIM) approaches into collaborative design workflows. Design data is primarily stored in discipline-specific BIM models, which users typically access as distinct monolithic files. An especially important application of model-based workflows is the validation of models against a predefined rule set, a process often referred to as BIM-based model checking (Eastman et al., 2009) These rules are derived from client requirements or codes and standards that the multidisciplinary design must comply with. Any identified problems are recorded as issues and returned to the designer, who is responsible for adjusting the design models so that they ultimately conform to the stated rules.

Model checking is typically performed at predefined milestones and on a complete model without reusing results from previous runs. The intermediate steps designers take to resolve issues are neither documented nor communicated. Consequently, current checking systems do not support incremental validation on updates, increasing checking time and resulting in unnecessary rework. So far, no mechanisms are in place to automatically track whether the changes have resolved the previously identified issues. As a result, engineers or reviewers must manually review the model-checking results to determine whether new issues need to be documented and assigned, and which existing issues were resolved in the latest design iteration.

Contribution

This paper introduces a new method that applies version control concepts to transmit only the changes made to a BIM model, rather than sending entire model files. These version increments are subsequently used to provide the rule system with more precise information about which modified model elements need to be checked. To this end, the graph transformations that constitute the version increments are supplemented with higher-level annotations that express the engineering operations performed and identify the affected parts of the BIM model. Consequently, only those rules influenced by recent model changes need to be re-executed when a model increment is shared. Furthermore, associating each version increment with its enriched descriptors and the rule outcomes generates a valuable information resource for the subsequent issue management, as it reveals which issues may need a status update and directs the engineer to areas that warrant particular attention.

Related Works

BIM-based Model Checking and Issue Processing

The fundamental concepts of BIM-based model checking were introduced by Eastman et al. (2009) and subsequently adopted and further developed by other scholars. In summary, a model-checking routine typically involves four main stages: translating the rules into a machine-readable language, preparing the building model data, carrying out the checking process, and finally presenting the results.
Lee et al. (2026) provides an overview of recent advances in code-compliance checking, highlighting three key research areas: interpreting regulatory text into computable formats, model checking workflows with BIM and IFC data, and integrated workflows linking interpretation, checking, and reporting. To overcome manual translations, Fuchs et al. (2024) employed Large Language Models (LLMs) to convert regulatory documents into actionable, machine-executable representations. The execution logic of the checking routines must also be presented clearly and transparently. Preidel and Borrmann (2016) proposed a visual programming framework that accelerates the creation of rules and provides more flexibility than conventional model-checking tools. As an extension to these previous works, Iversen and Huang (2026) have investigated whether LLMs can directly be used for rule

interpretation, information extraction from BIM models, rule execution, and reporting bypassing manual preparation of distinct elements in a model-checking workflow. Even though these advancements are promising, most software products on the market provide predefined rule templates that can be adapted to specific scenarios to be checked.

Despite these advances in rule interpretation and representation, existing automated compliance checking workflows still determine the scope of checking through model-level filtering. This mechanism presupposes that the entire BIM model is available and consistently structured, but more importantly, it prevents the system from identifying which elements are affected by design changes. As a result, the current approaches cannot support incremental compliance checking and instead require full re-evaluation after every update.

Design Intent and Rationale

Every decision made during the design and planning phase of a built asset entails reflections on *how* and *why* it is constructed. Although these modes of thinking are at the heart of the design and planning disciplines in the AEC sector, they are rarely explicitly documented in design models. Wyke and Munch Lindhard (2025) report that it would be highly advantageous if the models also contained the underlying engineering intent that caused a change, as well as the reasoning behind the modifications performed. Developing effective methods to capture the rationale of a design has been a longstanding goal in engineering design (Siddharth and Sarkar, 2018). Without an explicit representation of design intent, design changes can easily introduce inconsistencies in building planning. For example, changes in architectural design can cause sprinkler layouts to become misaligned, making MEP revisions necessary to reestablish the intended coordination with the updated architecture (Sacks et al., 2022).

Currently, issue-related communication for BIM models is primarily facilitated via the BIM Collaboration Format (BCF). Because the BCF provides only very limited metadata about identified issues, the actions taken to resolve them can currently be recorded only as written comments attached to the issue. Such textual notes can convey intent and rationale only to a very limited extent, which is insufficient to support further automation of downstream processes. Consequently, the engineering intent behind a model change is generally implicit and is expected to correspond to the issues and defects revealed in the preceding model check. To support the design decision-making process, Wu et al. (2025) introduce a Design Healing framework that searches for code-compliant alternatives to resolve identified building code violations while remaining close to the originally intended solution. This framework focuses on localized changes, such as wall movements, and avoids modifications that would alter the existing building design topology. However, design changes in practice often alter the original building topology, for example, by creating or deleting building elements, and there is still limited support for systematically analyzing the underlying engineering intent behind such changes. Hu et al. (2021) capture the dependency between clashes and generate an optimal component change list to minimize the impact on the project. This method addresses the challenge posed by highly interdependent building systems, where clashes can propagate across related components.

Change propagation and version control of BIM models and issue representations

To establish a connection between implemented changes and unresolved issues, a robust method is needed to record and reveal the modifications made to the design models. Precisely monitoring changes across multiple model versions is labor-intensive and has to be performed again for every new version. This situation was primarily driven by the lack of version control techniques for BIM models that track changes based on the underlying object structures in the past.

As a result, changes between different model states often require substantial manual investigation before potential impacts on other disciplines can be evaluated. Whereas some methods have focused solely on algorithms for performing diff operations directly on the underlying data structures of BIM models (Shi et al., 2018; Liu et al., 2026), Esser et al. (2022) proposed dedicated version control mechanisms grounded in graph-based representations of BIM models, enabling more fine-grained, efficient management of model changes. In essence, BIM models are transformed into Labeled Property Graphs. Changes are then classified as either topological modifications covering inserted or removed nodes and edges, or semantic modifications, capturing attribute-level edits to structurally unchanged nodes. Any change information is ultimately assembled into a graph transformation rule p_{ij}, which is exchanged as an incremental change description. Figure 1 illustrates the approach schematically.

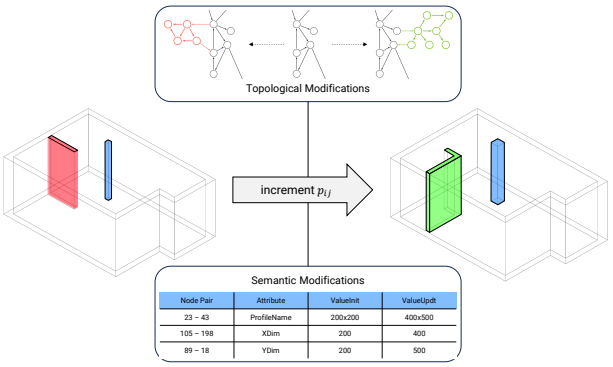

Figure 1: Incremental Version Control of BIM models.

Further efforts have focused on embedding version-control concepts directly into the issue representations. Oraskari et al. (2022) explored different ways to capture the history and evolution of issues within bcfOWL, which encodes issue data as a triple-based semantic graph in accordance

with the Resource Description Framework (RDF) principles.

There has also been research conducted to analyze the as-happened BIM authoring process using BIM logs. As data sources for understanding incremental authoring events, BIM logs provide information about how models are revised over time. Although BIM logs have initially been developed for monitoring and maintenance of authoring systems, several studies have enhanced their metadata to better capture modeling activities (Jang et al., 2023). Yarmohammadi and Castro-Lacouture (2018) introduced a bounding-box-based building element logs to quantify modeling performance. Gao et al. (2021) proposed a graph-structured event log that links modeling commands to the geometry they affect, enabling the analysis of trial-and-error patterns during authoring. Although these efforts commonly focus on improving the representation of element-level modeling events and revision trajectories, such logs do not capture the inter-element relationships that are essential for understanding model compliance. Moreover, they primarily record the state of BIM elements affected by modeling actions, whereas compliance checking requires tracking change propagation, which is significant in model checking.

Identified research gap

As discussed, model checking not only requires a carefully prepared set of rules but also well-prepared BIM models to be subjected to the checking process. However, current rule-checking workflows typically re-execute the entire rule set for each new design model file, regardless of what has changed in the interim, reflecting a lack of mechanisms to identify incremental checking targets. Moreover, existing approaches do not provide explicit representations to capture how changes in one model element propagate to related elements and their associated rules, making it difficult to reason about the true impact of a modification. Finally, there is no structured way to record compliance issues, the changes made in response, and how those issues are resolved over time. Incorporating version control principles into BIM-based workflows could address these gaps by systematically tracking edits, modeling their propagation across related elements and rules, and maintaining structured records that support more transparent, targeted, and repeatable compliance checking. Therefore, this paper aims to bridge these gaps by linking model updates to the subset of rules required to evaluate the performed changes, connecting checking results with subsequent issue information for coherent tracking across model updates, compliance assessments, and project communication items.

Proposed Methodology

Our approach aims to improve the preparation of the building model and the rule execution process by determining which rules to apply based on the exact changes made to the design model. When new components are first introduced to the design model, they are evaluated against the complete set of rules relevant to that modification scope. Any subsequent changes are then evaluated only against the subset of rules affected by those updates. In particular, by applying version control to the design models as they evolve, we gain direct access to detailed information about the changes made. The version increment that records these modifications describes the alterations at a strictly data-representation level following concepts of set and graph theory. Consequently, we propose additional enrichment and processing of this information to enable its effective use in the subsequent workflow.

Processing Pipeline

The envisioned processing pipeline is illustrated in Figure 2 and considers a designer who incrementally enhances their BIM model containing information about a specific discipline or trade, and a model checking workflow is used to validate whether the model complies with a given set of rules, indicated in the right part of Figure 2. The modification details captured in the version increment are enriched to determine which rules must be executed, enabling a more efficient assessment of the model's compliance. To accomplish this, four processing steps need to be incorporated into the standard model checking workflow:

Rule setup: Each rule performs checks on specific elements contained in the BIM model. In this way, we can understand which elements are checked by which rules and what additional context they require.

Increment Interpreter: In this stage of processing, we apply pattern matching to the version increments to derive engineering-level insights from the underlying graph transformation.

Assignment Module: This module uses the version increment and the identified modification scope to determine and trigger which rules must be re-validated in light of the design changes made.

Reasoning Module: In this step, the outcome of the performed check, along with the version increment, is used to decide whether a new issue must be created or whether satisfying the rule should instead trigger an update to an issue's status or its closure.

Rule setup

Figure 3 schematically shows the configuration of the rules. Each rule is defined by its application target, the necessary model context, a modification scope to which the rule is sensitive, and the specific checking procedure. Together, these elements form the digital representation of a particular aspect derived from a code or standard that the design is required to comply to.

The *application target* defines the primary focus of a rule, specifying which elements are explicitly examined. Targets can be selected based on element types and/or spatial or functional containment, or their combinations (e.g., all windows from the second floor upward). In addition to the application target, a rule can specify *supplementary model context*. This context provides additional informa-

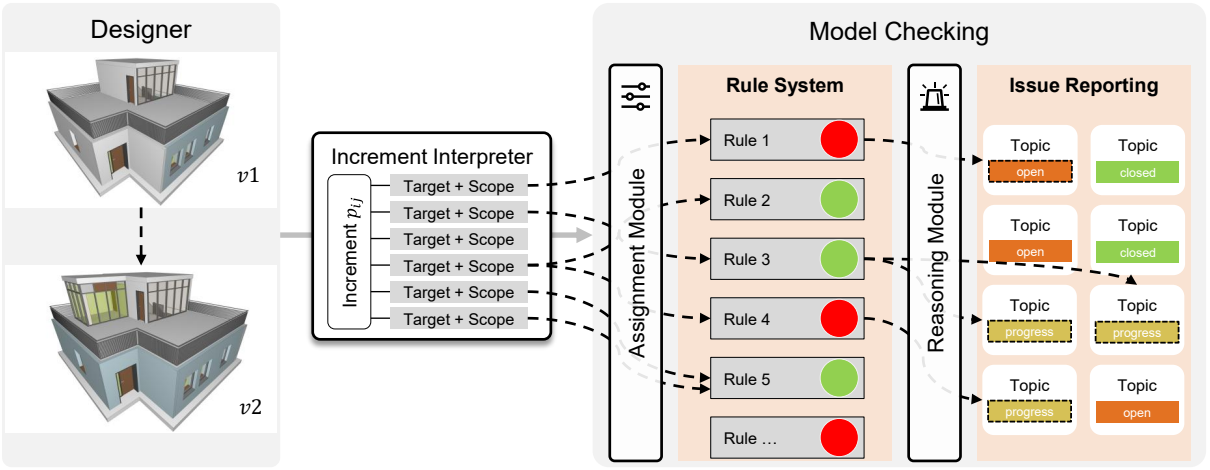

Figure 2: Overall processing pipeline: a model author creates two versions of a BIM model, from which version control principles produce an increment representing the performed changes. The increment is enriched before being fed into the model checking system.

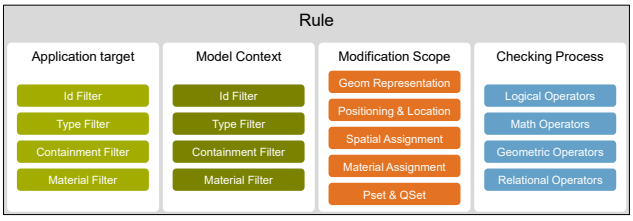

Figure 3: Rule Setup with the Application Target, Model Context, Modification Scope, and Checking Process.

tion that the checking process may require for execution. It can be retrieved from the model at any time, independently of whether the corresponding elements were modified in the latest design iteration.

For example, consider a rule that checks whether doors intersect with furnishings when opened. The application target of this rule would be the insertion of doors or modifications to their geometry and position. However, the furnishing elements are part of the model context and are fetched from the BIM model when the checking process needs them for geometric overlap calculation. Because rules can depend both on their application target and on their model context, rule execution must be defined in a commutative way: the rule must be re-executed not only when the application target changes, but also when elements forming the model context are modified — even if the application target itself would not otherwise be triggered.

The *modification scope* serves as a trigger interface that determines which types of model modifications the rule must be assessed for, both in terms of the application target and the model context. It consists of various categories of changes that can occur within a BIM model and is used to determine, for both the specific elements addressed by the rule (application target) and the surrounding conditions in the BIM environment (model context), under which circumstances the rule should be executed. The operators shown in the checking process section in Figure 3 correspond to those introduced by Preidel and Borrmann (2016), but can be expanded further to address specific issues.

Increment Interpreter

The version increment that captures the applied model modifications is formalized as a graph transformation rule and adheres to well-established notations from fundamental graph and set theory. In the method presented by Esser et al. (2022), a Double-Push-Out formalism is used to define how the underlying object network forming the BIM model must be altered by deleting and inserting subgraphs to transform the model from its initial state to its updated state. These subgraphs are described in the Left-Hand-Side (LHS), the Interface graph I, and the Right-Hand-Side (RHS), reflecting the required transformation steps.

The modification operations captured in a version increment describe changes at the raw data level, interacting only with nodes and edges in the graph. While model updates at any level of granularity can be reliably expressed in this way, additional interpretation is required to derive higher-level information from these version increments that can guide our decisions about which rules need to be re-evaluated. Consequently, the increment interpreter is introduced as a preprocessing step before the data enters the model checking system (as illustrated in Figure 2). Figure 4 illustrates the interpretation process.

In this processing step, we examine the contents of the version increment and perform pattern matching on the graphs that constitute the underlying transformation rule, describing the necessary actions to update the BIM model (i.e., its graph representation) from version 1 to version 2. More formally, this means that we look for an occurrence of an operation pattern O_i within the graph patterns that form the Left-Hand Side (LHS) and the Right-Hand Side (RHS) of the transformation rule. The operation patterns O are assumed to be predefined and are supplied as a library. For each pattern, the library provides additional logic that extracts information about the target of the modification (e.g., the globally unique identifier of the object

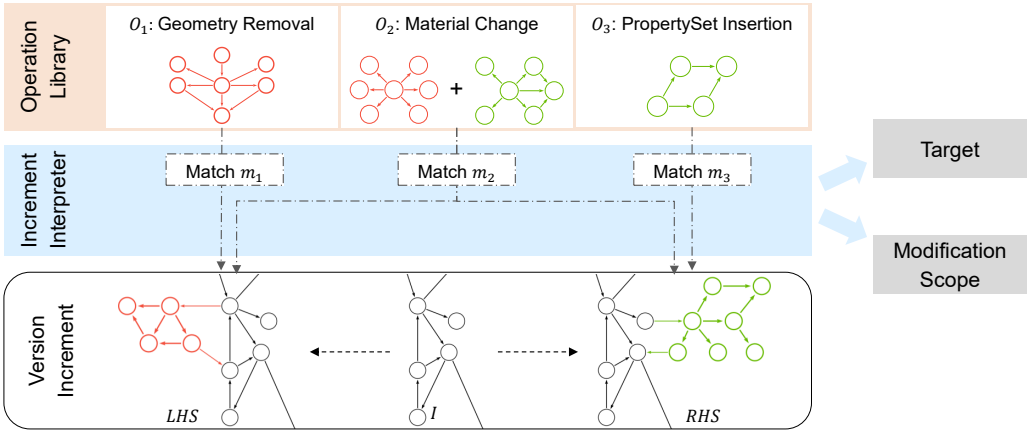

Figure 4: Increment interpreter: The version increment obtained from comparing two versions of a BIM model is shown at the bottom. The Interpreter's goal is to identify occurrences of the graph patterns defined in the operation library to reveal the scope and its target.

being modified) and the modification scope. Both aspects form the output of the increment interpreter, which will then be utilized in the assignment module.

Naturally, semantic changes also provide an informational foundation, meaning that the graph paths uniquely identifying nodes affected by semantic changes can likewise be used to infer higher-level statements about the target and the extent of the modification. Common examples of semantic changes include modifications to the Cartesian coordinates that define an element's location, as well as alterations to its name or description attributes, which are attached to nodes representing instances of subclasses derived from IfcRoot.

Assignment Module

The assignment module is responsible for linking the modification scopes and application targets, as determined by the Increment Interpreter, to the rules that must be executed in response to the detected model changes. In essence, it decides which rules are affected by which changes and orchestrates their re-execution. When existing components in the model are modified, the corresponding rules are selectively re-triggered, restricted to the identified modification scope, so that only the impacted portions of the model are reconsidered. In contrast, when elements are removed, any rules that reference them within the model context must be re-evaluated. If deleted elements were the original targets of specific rules and had already caused problems, those rules may be closed once the elements are removed. Similarly, when new elements are introduced, the assignment module identifies all rules that are applicable to these elements and ensures that they are executed accordingly. This targeted rule assignment enables incremental, efficient, and consistent evaluation of model changes without recomputing the entire rule set.

Reasoning Module

After the rules have been evaluated, their outcome is essentially a basic pass-or-fail result, typically accompanied by information about the elements that were checked and failure information about missing properties or violated parts of rules. These results are then further processed to support the system operator in managing issues.

In this step, the Reasoning Module preserves an explicit link between the performed model changes, the rules that were (re)evaluated as a consequence, and the issues that arise from those rule activations. It maintains the connection from the originating rule, through its evaluation outcome, to the corresponding problem descriptions and the resulting change requests addressed to the model authors. The issues themselves are recorded using the BIM Collaboration Format (BCF), which stores key metadata for each identified problem, such as its name, description, and the GUIDs of the affected elements. On top of this, the issues are enriched with identifiers that allow us to trace back precisely which rule triggered the documented problem or topic, and which model revision it relates to.

Because we can track how modifications to the model affect each rule's output, we can also determine which open issues are likely to be resolved by the considered model update. Although much of this analysis can be automated using the maintained links between model changes, rule activations, and issues, the final decision on each issue's status is ultimately left to the model reviewer.

Rationale of the Processing Pipeline

The proposed approach integrates into existing model-based workflows as a preprocessing layer, without requiring fundamental changes to the underlying rule system or issue management infrastructure, while the rule setup module enables practitioners to define checking targets at a finer granularity than conventional model-level filtering. The increment interpreter replaces the implicit assumption that reviewers can manually identify the scope of a modification, and the assignment module translates this information directly into rule-execution decisions, ensuring that only the affected subset of rules is re-triggered, thereby reducing the computational overhead associated with frequent model updates. Beyond efficiency gains, the reasoning module closes the feedback loop by connect-

ing rule outcomes to issues, introducing a level of process transparency that current workflows lack and enabling reviewers to trace why a particular check was triggered and which design decision prompted it. Over time, the accumulated change-issue pairs constitute a structured record of a project's compliance history, which could inform both design decisions and quality assurance processes in future projects.

Case Study

The following case study demonstrates the key ideas of the proposed method using a small-scale building model. The designer arranged a main hall as the central meeting space, with several auxiliary service rooms located at the rear of the building. To verify the design compliance, two specific rules out of a larger rule set are considered:

- Exit-Positioning rule: When two exits are required from an exit access area, they must be separated by at least one-half of the longest diagonal of the area, measured in a straight line [1].
- FireRatingPSet rule: Each door and window must be assigned a property set that includes its fire rating information.

Each rule provides the system with the parameters that it requires to run. For the Exit-Positioning rule, doors constitute the primary application target, and the rule requires the adjacent space geometry as model context in order to derive the minimum clearance distance to be maintained between individual doors. The rule is therefore sensitive to any modification in the geometric representation of the doors or their placement within the spatial configuration. In comparison, the FireRatingPSet rule applies specifically to doors and windows and focuses on fire-resistance data. Consequently, this rule reacts to modifications in the corresponding pSet attributes.

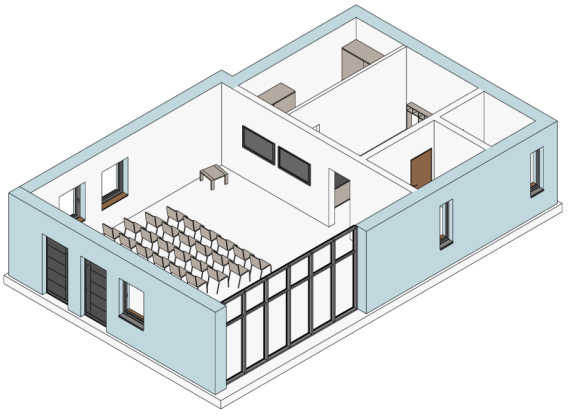

Figure 5: Case Study Model.

[1] Summarized from IBC Rule 1007.1.1, available `https://codes.iccsafe.org/content/IBC2018P6/chapter-10-means-of-egress#IBC2018P6_Ch10_Sec1007.1.1` (last access 2026-01-26)

Figure 5 illustrates the model in its initial version state, which is checked against any rule, as any information contained in the design model was newly inserted. In the initial design, two primary exit doors are positioned next to each other to provide access to the main hall. Given that the diagonal of this room is approximately 11.30 m, the spacing between these doors is clearly insufficient. Consequently, an issue is generated directly from the detected violation, and the association between the rule and the resulting issue is recorded. All doors and windows in the initial design contained adequate property set data; therefore, the FireRatingPSet rule confirmed their compliance. After returning the detected issues to the designer, they revised the model. Figure 6 illustrates the design modifications performed. One door has been repositioned on another wall, and a new window has been inserted.

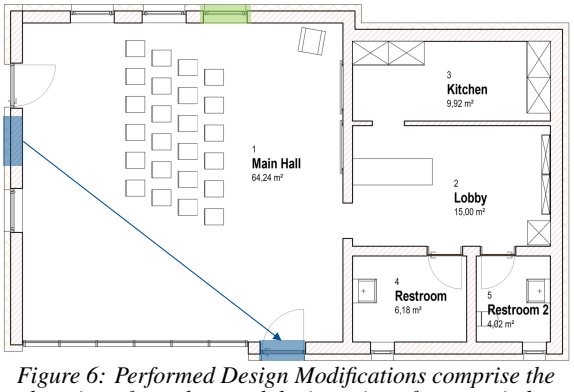

Figure 6: Performed Design Modifications comprise the relocation of one door and the insertion of a new window.

The changes are recorded as a version increment, comprising 287 nodes and 282 relationships in the removed pushout pattern, and 613 nodes and 637 relationships to be inserted via the inserting pushout pattern. Furthermore, 1817 node attributes have been edited. These nodes have not been altered in their topological position within the model graph but underwent semantic changes.

Figure 7 shows the operation patterns O that are available in the operation library. The increment interpreter applies these patterns to recognize that an entrance door has been modified and then reports the changes to the object's position and its nesting assignment, since the door is now placed on a different wall. As the version increment does not alter the fire resistance parameters of the relocated door, it is concluded that the door continues to comply with the fire resistance rule. Consequently, only the Exit-Positioning rule is re-assessed. As this rule reports no violations, the issue raised based on the original design model can be marked as resolved by the model modification and may be closed by the model reviewer, given the positive evaluation result. However, the newly inserted window is checked against the FireRatingPSet rule to ensure it meets the required element criteria. If compliance is confirmed, no issue needs to be created; if not, a new issue is generated that retains information about the originating rule and the non-compliant element.

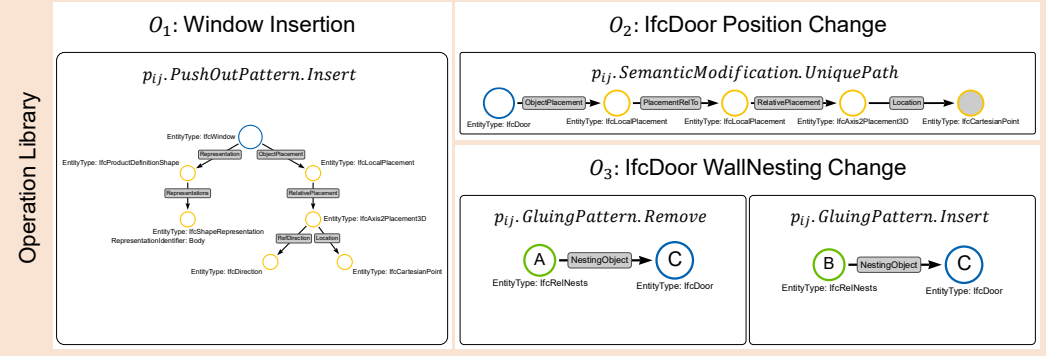

Figure 7: Case Study: Operation library used to identify the performed design modifications.

Limitations

Despite the potential benefits of the proposed approach, several limitations must be acknowledged.

First, the proposed concept relies heavily on an operation library that maps concrete design modifications to corresponding rule triggers. A core limitation is the assumption that this operation library is complete and accurate. In reality, constructing such a comprehensive operation library is challenging as certain modeling operations may not be fully captured or formalized. Furthermore, new modeling patterns and tools can introduce operations that are not yet known to the increment interpreter. If the operation library is incomplete or inaccurately specified, relevant triggers may not be fired, resulting in missing relevant checks. Consequently, the overall reliability of the approach is directly dependent on the continuous maintenance, validation, and extension of this library.

The current approach focuses on design changes that explicitly trigger rule checks. However, there can be intricate rules, which can be violated implicitly, even when no formal change trigger has been activated. Modifications in one part of the model may indirectly affect constraints or assumptions embedded elsewhere in the rule system, leading to undetected non-compliance. For example, design compliance with evacuation time depends on many interrelated factors such as space connectivity, door and corridor widths, and fire compartmentation. A local change, such as moving a door or partition, may not trigger an explicit evacuation check, yet it can still invalidate the relevant computed egress path or partially undermine its underlying design intent. While running the complete rule system on a periodic basis (e.g., weekly) partially mitigates this risk, it does not eliminate the possibility of temporary non-compliant states between full runs. In practice, this means that some violations may only be discovered with delay, and selective checks cannot yet guaranty full coverage of all rule interactions.

Another limitation of this study can be found in the limitation to single-discipline problems. Many industrial model-checking scenarios, however, involve multidisciplinary design models, where several domains are simultaneously represented. Interdisciplinary dependencies among building elements are common, and a change in one discipline may require corresponding updates in others to maintain conformance with relevant constraints and compliance requirements, for example, moving a wall may require relocating pipes and connected equipment in the MEP model. The current reasoning approach primarily targets change analysis within a single domain and must be extended to serve for interdisciplinary aspects, accordingly. In summary, while the method improves responsiveness and focuses rule-checking effort around actual changes, its effectiveness is constrained by the completeness of the operation library, the handling of implicit rule interactions, and the ability to reason over multidisciplinary models.

Summary and Outlook

In this paper, we introduced a novel approach to improving model-checking workflows by leveraging incremental version control as a fine-grained source of information for intelligent rule selection and subsequent reasoning about issue documentation. The approach decomposes the overall problem into smaller processing steps. First, version control principles are applied to track the evolving BIM model on the author's system. Performed changes are encoded as version increments, which are essentially graph transformations leveraging the underlying object network in the BIM model from its initial to an updated state. Then, we use these version increments to guide informed rule selection and focused model checking. Instead of running the full rule set after every modification, our method constructs a dependency mapping between model-edit operations and the rules that may be affected by those edits. Guided by this mapping, the model checking is invoked selectively, concentrating only on rules whose input parameters have been modified. By linking the outcomes of checks to the resulting issues, the approach enables both retrospective and predictive reasoning. Detected issues are associated with the specific changes that introduced or resolved them, thereby building a corpus of annotated change–issue pairs.

By exposing the relationship between detected issues and the underlying model changes, our method offers deeper insights into model evolution and the root causes of problems. Future work should extend the current illustra-

tive case study through experiments on real-world projects to examine more explicitly the reduction in rule executions enabled by enhanced traceability in model checking workflows. Moreover, collecting and learning from this data may enable future systems to propose solutions to issues with similar characteristics, paving the way for more proactive, intelligent support in model-based design environments.

Use of AI tools

While preparing this work, the authors utilized Large Language Models inside Overleaf Writefull AI to refine the language and identify spelling errors. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Acknowledgments

The research presented has been financially supported by the Federal Ministry of Education and Research (Bundesministeriums für Bildung und Forschung) in the frame of the project BauPuls360 under grant number 02K23A096. Furthermore, the work was supported by the funds of TUM Georg Nemetschek Institute. The responsibility for the content of this publication lies with the authors.

References

Eastman, C., Lee, J.-m., Jeong, Y.-s., and Lee, J.-k. (2009). Automatic rule-based checking of building designs. Automation in Construction, 18(8):1011–1033. doi: 10.1016/j.autcon.2009.07.002.

Esser, S., Vilgertshofer, S., and Borrmann, A. (2022). Graph-based version control for asynchronous BIM collaboration. Advanced Engineering Informatics, 53:101664. doi: 10.1016/j.aei.2022.101664.

Fuchs, S., Witbrock, M., Dimyadi, J., and Amor, R. (2024). Using Large Language Models for the Interpretation of Building Regulations. Journal of Engineering, Project, and Production Management, 14(4). doi: 10.32738/JEPPM-2024-0035.

Gao, W., Wu, C., Huang, W., Lin, B., and Su, X. (2021). A data structure for studying 3D modeling design behavior based on event logs. Automation in Construction, 132:103967. doi: 10.1016/j.autcon.2021.103967.

Hu, Y., Castro-Lacouture, D., Eastman, C. M., and Navathe, S. B. (2021). Component Change List Prediction for BIM-Based Clash Resolution from a Graph Perspective. Journal of Construction Engineering and Management, 147. doi: 10.1061/(asce)co.1943-7862.0002092.

Iversen, O. and Huang, L. (2026). Leveraging large language models for BIM-based automated compliance checking. Automation in Construction, 182:106707. doi: 10.1016/j.autcon.2025.106707.

Jang, S., Lee, G., Shin, S., and Roh, H. (2023). Lexicon-based content analysis of BIM logs for diverse BIM log mining use cases. Advanced Engineering Informatics, 57:102079. doi: 10.1016/j.aei.2023.102079.

Lee, J. H., Ji, S. Y., and Ostwald, M. J. (2026). Automated compliance checking across the building lifecycle: Systematic and semantic review integrating PRISMA and deep search. Automation in Construction, 185:106859. doi: 10.1016/j.autcon.2026.106859.

Liu, H., Ma, J., Gao, G., and Gu, M. (2026). Decoupling IFC models for reliable modification of shared references. Automation in Construction, 181:106670. doi: 10.1016/j.autcon.2025.106670.

Oraskari, J., Schulz, O., and Beetz, J. (2022). Towards describing version history of bcf data in the semantic web. In LDAC 2022: 10th Linked Data in Architecture and Construction Workshop, page 87.

Preidel, C. and Borrmann, A. (2016). Towards Code Compliance Checking on the Basis of a Visual Programming Language. Journal of Information Technology in Construction, 21:402–421.

Sacks, R., Wang, Z., Ouyang, B., Utkucu, D., and Chen, S. (2022). Toward artificially intelligent cloud-based building information modelling for collaborative multidisciplinary design. Advanced Engineering Informatics, 53:101711. doi: 10.1016/j.aei.2022.101711.

Shi, X., Liu, Y. S., Gao, G., Gu, M., and Li, H. (2018). IFCdiff: A content-based automatic comparison approach for IFC files. Automation in Construction, 86(June 2016):53–68. doi: 10.1016/j.autcon.2017.10.013.

Siddharth, L. and Sarkar, P. (2018). A Multiple-Domain Matrix Support to Capture Rationale for Engineering Design Changes. Journal of Computing and Information Science in Engineering, 18(2). doi: 10.1115/1.4039850.

Wu, J., Nousias, S., and Borrmann, A. (2025). Design Healing framework for automated code compliance. Automation in Construction, 171:106004. doi: 10.1016/j.autcon.2025.106004.

Wyke, S. and Munch Lindhard, S. (2025). Understanding Design Rationale and Intent through Natural Language Processing analysis: A search for Consensus. Journal of Information Technology in Construction, 30:631–649. doi: 10.36680/j.itcon.2025.026.

Yarmohammadi, S. and Castro-Lacouture, D. (2018). Automated performance measurement for 3D building modeling decisions. Automation in Construction, 93:91–111. doi: 10.1016/j.autcon.2018.05.011.