



# Incremental model updates in BIM: a critical analysis of the collaboration platform Speckle

Kai Dietmair<sup>1</sup>, Sebastian Esser<sup>1</sup>  and Benedict Harder<sup>1</sup> 

<sup>1</sup>Chair of Computational Modeling and Simulation, Technical University of Munich, Arcisstraße 21, 80333 Munich, Germany

E-mail(s): kai.dietmair@tum.de, sebastian.esser@tum.de, benedict.harder@tum.de

**Abstract:** Current practice and international standards implement BIM collaboration based on domain model federation, which leads to repetitive exchange of model versions as monolithic files. Subsequently, stakeholders receiving new model versions must manually check them to identify potential impacts on their own disciplines, which is labor-intensive and error-prone. These limitations can be overcome by implementing incremental update procedures. This way, only the information about the modified objects is transmitted in the event of model changes. In this paper, we examine the collaboration platform Speckle and critically assess its approach to implementing incremental model updates. Speckle provides an open and vendor-neutral data format and is known for its extensibility. We address several issues that were identified in Speckle's hierarchical data structure concerning the insertion and removal of model objects. An improved data structure inspired by Labeled Property Graphs will be presented, enabling more intuitive updates of BIM models and facilitating more accurate version control by maintaining object relationships.

**Keywords:** Speckle, incremental model updates, version control, BIG Open BIM



Erschienen in Tagungsband 35. Forum Bauinformatik 2024, Hamburg, Deutschland, DOI: will/be/assigned/by/editor  
© 2024 Das Copyright für diesen Beitrag liegt bei den Autoren. Verwendung erlaubt unter Creative Commons Lizenz Namensnennung 4.0 International.

## 1 Introduction

The planning of the built environment is characterized by a large number of involved engineers from different disciplines. Collaboration and communication are essential to achieve an overall consistent design. In the last two decades, the traditional exchange of information in the form of printed plans and documents has increasingly been replaced by model-based approaches from the field of Building Information Modeling (BIM) [1]. A BIM model contains geometric and semantic information about a building and can be encoded in various file formats [2]. Due to the iterative nature of planning processes, models are often subject to changes after they have already been exchanged with other project participants. Today's common practice is to share the modified model as a new monolithic file with other stakeholders. They have to check the new model for any changes applied and to identify inconsistencies with their own models, which is labor-intensive and error-prone. This situation can be improved by using incremental update procedures, in which only the information about the modified objects is transferred in the event of a change.

In this paper, we critically assess the approach pursued by the collaboration Platform Speckle [3] when incrementally updating BIM models. To this end, the design of the Speckle platform is summarized, and case studies are conducted to evaluate its power. Subsequently, we address several issues concerning the insertion and deletion of model objects that were identified in Speckle's data structure and present an improved structure inspired by graph-based methodologies. We aim to explore how Speckle's hierarchical data structure can be enhanced to address current limitations, ultimately providing more robust and intuitive version control for BIM models.

The paper is organized as follows. Section 1 presents our motivation and the problem statement. Section 2 briefly summarizes related work. Section 3 critically reviews Speckle's current data model, identifies issues within it, and proposes a solution to the identified issues afterward. Section 4 discusses the proposed approach and lists its advantages and limitations compared to the traditional hierarchic way of structuring increments. Section 5 concludes this paper and outlines potential directions for future research.

## 2 Related Works

Version control and associated techniques to transfer incremental changes have been subject to a wide variety of investigations in different industries. Very prominent examples are version control systems for source code, which are in common use today. In contrast to purely textual representations, however, BIM models appear as highly networked structures that contain information about objects and relations across these objects. Such interconnected structures are required to reflect the complexity of the built environment in a sufficient computer-interpretable representation. Over the past ten years, several concepts have been developed that address the challenge of version control for BIM models. Back in 2005, Firmenich, Koch, Richter, *et al.* [4] developed an approach that took up the concept of textual version control, which was already established in software development at the time and was intended to replace the exchange of entire files in the event of planning changes. They developed a procedure for serializing object sets and storing them in a text-based Version Control System (VCS). This allowed them to make use of existing VCS functions.

Some of the ideas of that time have been taken up by Speckle [3]. Speckle arose from the need of large planning and engineering offices for the coupling of models for data integration between specialist applications [5]. Speckle is an open-source vendor-neutral collaboration platform for planning the built environment, designed to facilitate collaboration between the various parties involved in a construction project. It was originally developed in 2016 at University College London by Dimitrie Stefanescu as part of the InnoChain project [6]. Speckle's main objectives are version control, creating compatibility between different domain-specific software products, storing authorship, and minimizing the amount of data to be transferred. The latter is realized by transferring only the modified model components in case of a model change and structuring them hierarchically.

In addition to Speckle there are also other open and vendor-neutral exchange formats for BIM models. One of them is represented by Industry Foundation Classes (IFC) [7], which is already widely supported by market-leading BIM software vendors. In this context, Shi, Liu, Gao, *et al.* [8] proposed IFCdiff, a content-based automatic comparison approach for IFC files based on a similarity metric. Unlike

traditional methods for comparing IFC files that rely on manual inspection or simplistic attribute checks, IFCdiff's approach reduces computational time and space requirements. Thus making it suitable for incremental backup of BIM data. IFCdiff constructs hierarchical structures that eliminate redundant instances for each of the two IFC files to be compared. Afterward, it calculates a similarity score using a recursive depth-first search. However, the calculated score is a mere scalar value that does not provide a deeper understanding of the actual changes applied to the model.

Esser, Vilgertshofer, and Borrmann [9] have developed an alternative approach to enable incremental updates of BIM models. Their central idea is to represent BIM models in the form of Labeled Property Graphs (LPG), which aim to represent complex and interconnected object structures better than text-based approaches. Similar ideas are also documented in Zhu, Wu, and Lei [10] or Ismail, Nahar, and Scherer [11]. Changes applied to a model are determined based on the graph representations of two IFC model versions. Subsequently, the derived changes are stored as graph transformation rules and exchanged with other project stakeholders. They can then apply the transformation to its outdated version of the graph and thus end up with the most recent model version.

With several incremental update methods for BIM models, the question arises as to which of them is the most suitable for today's construction practice and what the advantages and limitations of the respective methods are. In this paper, we critically look at Speckle's incremental update approach for BIM models and explore alternative ways of organizing the increments to be transferred. We draw inspiration from the graph-based approach in [9] to make suggestions for improving the hierarchic Speckle approach.

### 3 Critical Assessment

In this section, after a brief summary of Speckle's basic approach to incremental updates, we will identify problems in Speckle's way of organizing the increments to be transferred. Subsequently, we will present an approach to resolve the identified problems.

#### 3.1 Speckle's Data Model

Speckle is an object-based platform, meaning that data is not stored as monolithic files but as individual objects. The platform can be divided into two main components: the server and the connectors. The connectors are plug-ins for popular software products in the Architecture, Engineering, and Construction (AEC) industry. As of June 2024, Speckle offers 28 different connectors. When sending objects to the Speckle server, the connectors create so-called commits, which are essentially snapshots of the selected design data at a specific point in time. Each time objects are sent to the server, the corresponding connector converts the objects from their original software-specific format into Speckle's software-neutral format and serializes them into JSON format. When receiving objects from the server, the same conversion takes place in the opposite direction. This enables data to be transferred between different software products.

When a Speckle object is created, the corresponding connector assigns it a unique Speckle ID. This Speckle ID is then mapped to the object's native ID in the application where it was created. The native IDs of all new objects created when receiving the Speckle object are also mapped to this Speckle ID. This mapping can be stored in various repositories, such as the Speckle server or SQLite databases.

When receiving a commit from the server, the connector compares the native IDs mapped to the received Speckle objects with the native IDs of the objects currently present in the model and updates the native objects accordingly. This ID-based comparison is Speckle's basic approach to incremental updates.

Collections are the most generic objects in Speckle and are used universally by all connectors to hierarchically structure the objects contained in a commit. Collections possess an attribute called "elements", which represents a container for other Speckle objects. This means collections can contain further collections, leading to a nesting of Speckle objects. For example, the Revit connector automatically creates collections for all objects according to their Revit categories. Like collections, built objects are structured hierarchically in Speckle. Hosted Objects, such as doors or windows, are elements of their host objects and are, therefore, part of the host object's collection when committed to the server. Hosted objects are only assigned to a separate collection if their corresponding host objects are not included in the commit. An example case of this concept, where walls are used as host objects and doors as hosted objects, is depicted in fig. 1 and fig. 2. The BIM model used in this example was created within the BIM authoring software Autodesk Revit (version 2024.1). Selected model objects were then included in commits and sent to the Speckle server using the Revit Connector (version 2.17.0).

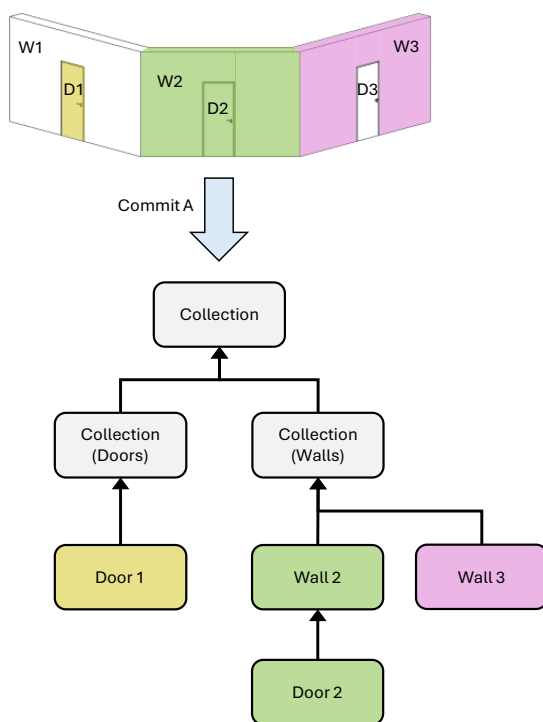


Figure 1: Commit A

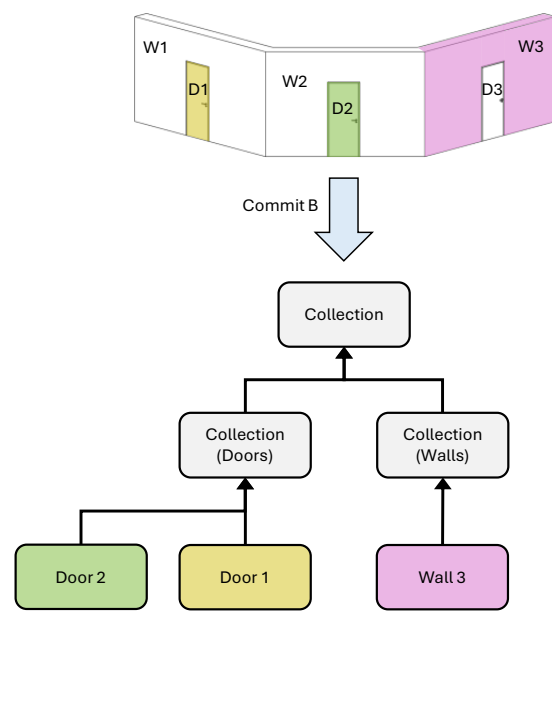


Figure 2: Commit B

### 3.2 Problem Identification

In many cases, Speckle successfully implements incremental updates of BIM models, especially if the transferred increments consist of moving and dimensioning existing model objects [12]. However, particularly during the insertion and deletion of objects hosted by other objects, the Speckle connectors

frequently encounter issues or fail to produce the expected results. As a cause for these issues, we identified a data model problem and additional requirements with the hierarchical structure of built objects and the strict separation of collections described in section 3.1.

The Revit connector creates commits in which host objects reference their elements via corresponding IDs. However, referencing in the opposite direction from the hosted object to the host object does not occur. For example, an element in the door collection cannot access its host wall. As a result, a newly created door committed as a single increment is not inserted into the model when received back from the server. Instead, to successfully insert the door, its host wall must also be included in the commit, which is not intuitive for the end user (cf. fig. 3).

The limitations of Speckle's current data model are apparent not only when inserting hosted objects but also when deleting objects. A Speckle commit can only contain objects. It is impossible to commit a "delete command" to remove an object when pulling the commit into another version of the model. Instead, to successfully remove an object from a Revit model, a commit must be received that contains at least two objects. If, for example, a door is to be removed from a model, it is not enough to simply receive its empty host wall into the model. Instead, the commit must contain at least one object besides the wall, such as a window (cf. fig. 4).

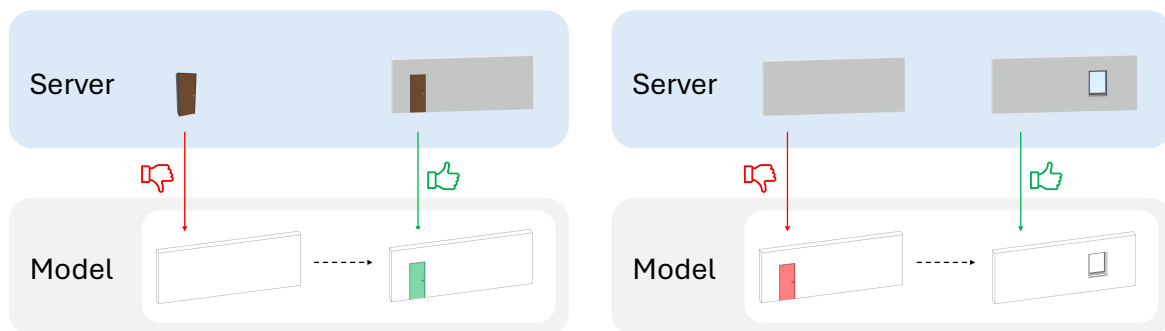


Figure 3: Inserting a door via Revit connector    Figure 4: Deleting a door via Revit connector

### 3.3 Proposal to overcome identified issues

This section presents the developed solution to the problem specified in section 3.2. The central idea is to abolish the hierarchical structuring of the built objects and the strict separation of collections from each other. Instead, the individual Speckle objects are now sorted into homogeneous collections, which only contain objects that actually belong to the same component category. For example, a door that is committed together with its host wall is no longer regarded as an element in the wall collection but as a separate object in the door collection. However, these homogeneous collections are not strictly separated from each other, but the individual Speckle objects they contain can be linked to each other. Here we fall back on the concepts used by Esser, Vilgertshofer, and Borrmann [9] when creating LPGs. The objects and collections are represented as nodes, and the connections between the individual nodes as labeled directed edges. A strict hierarchy between objects, therefore, no longer exists, as these edges are also formed across collections. For example, hosted objects reference their respective hosts via directed inter-collectional edges labeled as "fills". As a result, the collections and objects contained in a Speckle commit no longer form a hierarchical tree structure but a graph

structure. Another important component of the proposed solution is the virtuality of nodes and edges. Even if a hosted object is committed without its host object, the edge between the two persists virtually by connecting the hosted object to a virtual instance of its host. An example case of this concept is depicted as a Graph Meta Model in fig. 5 and fig. 6, taking up the example for Speckle's commit structure shown in fig. 1 and fig. 2.

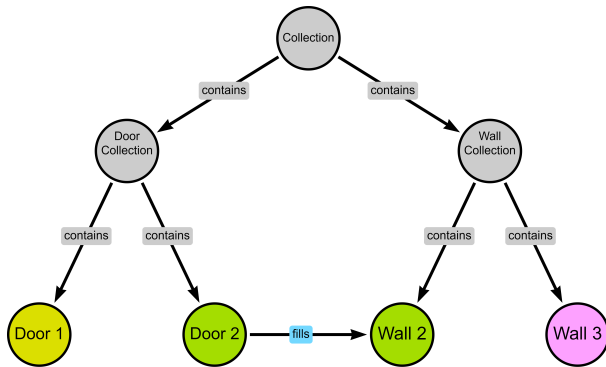


Figure 5: Commit A Graph proposal

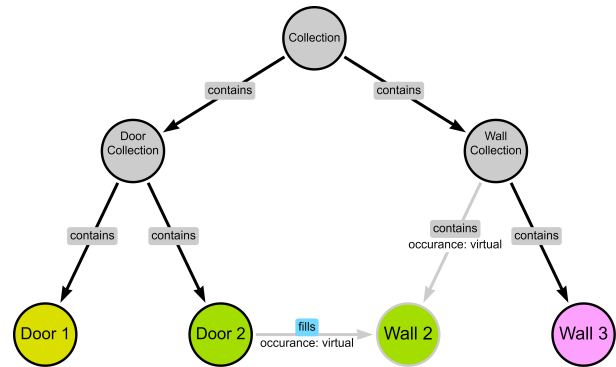


Figure 6: Commit B Graph Proposal

With this method, a newly inserted hosted object can be intuitively committed as a single object, retaining its reference to the host object. As a result, it can be inserted into all previous model versions containing the host object by receiving the commit back from the server. This way, the problem with inserting hosted objects mentioned in section 3.2 is resolved.

The proposed approach also improves the way objects are incrementally deleted from BIM models when receiving a Speckle commit. If a user removes objects hosted by another object, this former host object has no "fills" relationships when it is committed to the server. Therefore, all previously hosted objects are deleted when the former host object is received into an old model version. This resolves the issue of a commit containing only one object not being able to delete objects from a BIM model. The proposal could be implemented into Speckle's existing system by adding an additional attribute to the JSON serialization of hosted objects. By storing the IDs of the host objects in attributes of the corresponding hosted objects as references, it is possible to integrate the graph-based approach seamlessly into Speckle's current way of transferring increments. This modification would ensure that hosted objects retain their references to host objects, facilitating accurate and intuitive incremental updates.

## 4 Discussion

The hierarchical structuring of objects and collections offers a straightforward way to incrementally update BIM models, especially when using hierarchical exchange formats, such as JSON. However, our investigation identifies drawbacks in Speckle's hierarchical method, particularly in handling the insertion and deletion of model objects. The proposed solution, which involves transitioning from a hierarchical tree-like structure to a graph-based structure, offers a promising alternative. By leveraging the concepts of LPGs, our approach allows for more flexible and robust handling of object relationships. Homogeneous collections and directed inter-collection edges facilitate better representation of

object dependencies and ensure that hosted objects maintain their references even when committed independently from their respective hosts.

However, the proposed concept of virtual nodes and edges has its limitations. While it addresses the issue of incremental deletions and insertions of hosted objects by allowing the representation of objects not explicitly included in the commit, it also increases the amount of data that needs to be transferred. The goal should be to keep the transferred increments as small as possible and only as large as necessary. This trade-off between maintaining accurate object relationships and minimizing data transfer must be carefully considered in practical applications. Furthermore, the implementation of the proposed solution has not been realized yet. However, Speckle's architecture can be extended to meet such requirements.

## 5 Conclusion and Outlook

We have critically evaluated some of the Speckle functions currently available in June 2024. Issues about the separation and hierarchical structure of collections have been identified as presented in section 3.2. The proposed improved data model from section 3.3 addresses these issues by transitioning from a hierarchical tree-like structure to a graph-based structure with homogenous collections connected by inter-collection edges. Thus, the flexibility and robustness of object relationships when transferring incremental updates of BIM models are enhanced, ultimately improving the user experience and efficiency in managing construction projects. A preliminary solution for implementing the proposed concepts is to add ID-based attributes to the hierarchical serialization of model objects. However, further research and development are needed to realize the potential of the proposed solution fully. Future work could focus on the practical implementation and testing of the proposed model within Speckle and other BIM platforms. This includes evaluating the performance and scalability of the approach in real-world projects, as well as exploring additional features and optimizations that could further enhance its utility.

Overall, the proposed graph-based approach could significantly enhance Speckle's incremental update capabilities, addressing the identified issues and providing a more flexible, intuitive, and efficient data management framework for the AEC industry.



## References

- [1] S.-E. Schapke, J. Beetz, M. König, C. Koch, and A. Borrmann, “Prinzipien und Techniken der modellgestützten Zusammenarbeit”, in *Building Information Modeling*, Springer, 2021, pp. 309–333, ISBN: 978-3-658-33361-4. DOI: 10.1007/978-3-658-33361-4\_15.
- [2] M. Oh, J. Lee, S. W. Hong, and Y. Jeong, “Integrated system for BIM-based collaborative design”, in *Automation in Construction*, vol. 58, 2015, pp. 196–206. DOI: 10.1016/j.autcon.2015.07.015.
- [3] *Speckle*, 2024. [Online]. Available: <https://speckle.systems/> (visited on 02/07/2024).
- [4] B. Firmenich, C. Koch, T. Richter, and D. Beer, “Versioning structured object sets using text based Version Control Systems”, in *Proceedings of the 22nd CIB-W78 Conference on Information Technology in Construction*, 2005, pp. 105–112.
- [5] D. Mondino, “BIM im architektonischen Entwurf”, in *Building Information Modeling*, Springer, 2021, pp. 381–391, ISBN: 978-3-658-33361-4. DOI: 10.1007/978-3-658-33361-4\_19.
- [6] P. Poinet, D. Stefanescu, and E. Papadonikolaki, “Collaborative Workflows and Version Control Through Open-Source and Distributed Common Data Environment”, in *Proceedings of the 18th International Conference on Computing in Civil and Building Engineering*, 2020, pp. 228–247.
- [7] *BuildingSMART, Industry Foundation Classes (IFC)*, 2024. [Online]. Available: <https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/> (visited on 04/07/2024).
- [8] X. Shi, Y.-S. Liu, G. Gao, M. Gu, and H. Li, “IFCdiff: A content-based automatic comparison approach for IFC files”, in *Automation in Construction*, vol. 86, 2018, pp. 53–68. DOI: 10.1016/j.autcon.2017.10.013.
- [9] S. Esser, S. Vilgertshofer, and A. Borrmann, “Graph-based version control for asynchronous BIM collaboration”, in *Advanced Engineering Informatics*, vol. 53, 2022, p. 101 664. DOI: 10.1016/j.aei.2022.101664.
- [10] J. Zhu, P. Wu, and X. Lei, “IFC-graph for facilitating building information access and query”, in *Automation in Construction*, vol. 148, 2023, p. 104 778. DOI: 10.1016/j.autcon.2023.104778.
- [11] A. Ismail, A. Nahar, and R. Scherer, “Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard”, in *24th International Workshop on Intelligent Computing in Engineering (EG-ICE 2017)*, 2017.
- [12] K. Dietmair, “Analyse der Kollaborationsplattform Speckle im Hinblick auf inkrementelle Updates von BIM-Modellen”, Bachelor’s thesis, Technische Universität München, 2024.