Graph-based version control for asynchronous BIM level 3 collaboration

Sebastian Esser, Simon Vilgertshofer, and André Borrmann
Technical University of Munich, Germany
sebastian.esser@tum.de , simon.vilgertshofer@tum.de , andre.borrmann@tum.de

Abstract. Collaboration and communication are two essential aspects of Building Information Modeling (BIM). Current standards such as ISO 19650 take this into account by propagating the concept of federated domain models based on file-based information containers (BIM level 2). In consequence, complete models are transmitted every time a new version is shared with the collaborators. As changes in domain models cannot be tracked for individual objects, but for whole files only, high effort for the subsequent coordination across the domains is created. These limitations can be overcome by implementing modern approaches of digital collaboration based on object-level synchronization, as denoted as BIM level 3. To provide a methodological basis, this paper proposes to represent the object-networks of BIM models as formal graphs and describing changes in the model as graph transformations. Consequently, modifications can be transmitted as patches using the graph formalisms, which are to be integrated and interpreted on the receiving side, thus achieving object-level synchronization. The paper discusses in detail the graph-based representation and the implementation of the necessary graph comparison algorithms.

1. Introduction

Collaboration in projects of any size gain increasing importance in the AEC industry. Data exchange across experts of different domains and roles is one of the key aspects of Building Information Modeling (BIM). The degree of support for vendor-neutral data exchange formats by BIM-based software applications has increased during the past years and eases data handover between stakeholders.

Current practice for model-based collaboration, reflected by international standards such as ISO 19650, relies on the concept of federating disciplinary models in a common data environment (CDE) based on so-called information containers. As these information containers are basically a collection of files, the currently implemented mechanisms for model-based collaboration rely on mere file management, where files are the smallest manageable information unit (Preidel *et al.*, 2018).

In consequence, the complete domain model is transferred as monolithic file, each time a new version is made available. While these updates are very frequent during the collaborative design phase, it requires the manual identification of design changes by all other stakeholders. At the same time, the ratio between modified objects and the total number of objects in an updated model is often rather small. Therefore, providing the entire modified model is inefficient if other project participants have already received, understood, and integrated the foreign but outdated model version in their respective software environments.

To overcome the described limitations, improved techniques are required to enable the versioning of BIM models. This versioning includes identifying updates in models and transmitting solely the update information instead of the entire models. The communication between project participants is consequently realized by update patches that represent the

update procedure. To this end, a specific focus is put on possible mechanisms to detect changes and integrate update patches in the receiving application.

1.1 Outline

The paper introduces a novel approach that extends file-based collaboration to object-based collaboration using patch-based update mechanisms based on graph formalisms. The entire communication process can be split into three major parts: (i) the update identification, (ii) the patch formulation and distribution, and (iii) the patch integration on the receiver side. The information provided by (disciplinary) BIM models is represented by graph structures, which provide a well-established formalism in data science.

1.2 Preliminary Remarks

The term "model" is broadly used with widely diverging semantics in research and practice and can refer to various structures. The Meta Object Facility (MOF) specifications standardized by the Object Management Group (OMG) distinguish between instance data (M0), data model (M1), meta model (M2) and meta meta model (M3) (Object Management Group, 2019).

By contrast, in the BIM domain the term *model* often refers to the population of instance data. In the context of this paper, we accordingly use the term *BIM model* or *domain model* in the sense of MOF level M0. In addition, the underlying structure, which abstracts the given real-world problem from a certain perspective, is defined as data model or schema specification. The abstraction of a data model in its generic items like datatypes and relationships is defined in a meta model.

2. Background and related work

The increasing growth of digital technologies in the AEC sector has provided industry with opportunities to improve its productivity and operations. A central aspect is the improved communication and collaboration among contractors, coordinators, architects, and engineers. This is accompanied by the need to provide various structures for the transmission of information.

Versioning of structured data representations raises awareness in many industry branches for a long time now. Specifically, in the field of software development, various methods, protocols, and systems exist that enable distributed version control of text files. Prominent examples are Subversion, Mercurial and Git among others. In most approaches, a central database stores the global history of change events, integrates incoming modifications ("commit and push") and allows a user to clone the entire history with all incremental changes to his local machine. Therefore, each user can read and understand the entire history, create, and test modifications locally. If changes are ready to share with others, the user synchronizes his local state with the central database again. The chain of update messages forms the entire history of the project. Incoming updates can be integrated automatically if they do not create any conflicts with existing or concurrent local changes. Only in case of conflicts, the user needs to resolve them and choose the desired content manually (Blischak, Davenport and Wilson, 2016). In the context of this paper, we take inspiration from these version control systems but do not apply their principles on text files, but on graphs.

Existing versioning services use a line-based data comparison and track text lines that have been added, deleted, or modified. Data models used in the AEC-Sector, however, describe complex and highly interconnected information structures that cannot be versioned by a pure

text-based approach. For example, the order of entities might be completely different in two versions of a STEP physical file (SPF), regardless that the exact same information content is provided in both versions. Despite these limitations, text-based serialisations of data models are highly used to transfer BIM data in file-based handover scenarios. Looking into current practice in AEC projects, collaboration is mainly realized by means of file-based data exchange (BIM Level 2 according to ISO°19650). Actors from various domains work together using a central database, which is denoted as Common Data Environment (CDE) (DIN, 2019). CDEs help to share and coordinate domain models among involved actors. However, these platforms do not yet offer tools to realize object-level collaboration (BIM level 3).

To overcome the lack of applying object-level versioning in AEC projects, a clear understanding of common principles is necessary, which are used to define data exchange structures. Data models help to describe knowledge of a specific domain and can target various use cases (Turk, 2001). The data model itself is formulated in a schema definition, which defines a skeleton for the piece of information that needs to be exchanged. These skeletons follow mainly the principles of object-oriented programming paradigms. A class defines the frame (i.e., blueprint) on how an information gets stored using attributes and associations. Attributes have a name and a datatype. Associations point to other classes. Furthermore, a class can have one or many relationships to other classes. An instance of a class (an object) fills the given structure with specific values to describe the actual information the user wants to store and exchange. The associations between the instances result in an in-memory graph-like structure, also denoted as object network.

To exchange data stored in such in-memory class instances, an export module serializes the structured information into a file-based representation. These files are often in the ANSI-format and can span several thousands of text lines even for a relatively small scenario. As the term *serialization* implies, a sequential ordering of information is introduced even if the object network does not provide any kind of order. Therefore, text-based versioning systems will fail to correctly identify the modification in the underlying object graph, leading for example to the erroneous detection of massive changes for identical models when the serialization order is changed. Therefore, there is a need to improve the modification detection, which can reflect class instances and their relationships better than in a pure text-based versioning. Principles of graphs and graph transformation appear to be a promising approach to overcome the presented limitations. Graphs are a well-established concept to describe sets of nodes and their relationships among each other.

The application of graph-based systems for information management is not a novel approach in software applications. Many approaches in this field use the term *Graph Data Models* (GDM), which got introduced by Hidders (2001). The essential idea is that each class instance is represented as a node in a graph. Attributes are attached to a node whereas references or associations are represented by edges. Furthermore, graph structures and graph synthesis were successfully applied for information synthesis in other industries (Helms and Shea, 2012).

In the context of model analysis, several publications have investigated the application of graph analysis in recent years. Both, Tauscher, Bargstädt and Smarsly, (2016) and Ismail *et al.*, (2018) have explored graph-based representations of BIM models to navigate and query the object structure. Even though applying graph systems has been applied for various use cases, none of them tackle the problem of versioning model contents in a generic manner. Several established BIM applications expose methods to compare two IFC models (BIM Vision, 2021). These implementations, however, often base on suitable assumptions such as remaining GUIDs through the model versions, but do not capture any possible modification type applied to a model revision.

Shi *et al.* (2018) have proposed an approach that allows detecting differences between two IFC models based on a similarity metric. Their system runs a normalization on all instances stored in the model first and calculates a similarity score afterwards using a recursive depth-first search. A downside, however, is that the resulting similarity rate is presented a mere scalar value. Such score does not expose any kind of understanding of the actual change applied to the model.

3. Proposed framework and approach

The conducted literature has proven the need for versioning systems, explicitly targeting highly structured object-oriented data described according to schema specifications. As largely data models (formats) are currently used for vendor-neutral data exchange in AEC projects, the proposed concept is schema-independent, i.e., it supports diverse schemas if they follow a given set of boundary conditions. Simultaneously, it is not intended to create an entirely new data model that suits any possible use case but rather to keep the structures of existing and well-established standards. This approach acknowledges the development of exchange standards like the Industry Foundation Classes (IFC), RailML, and many others. We address the issue of version control in a generic manner by defining a generic graph meta model. Figure 1 denotes the overall data flow.

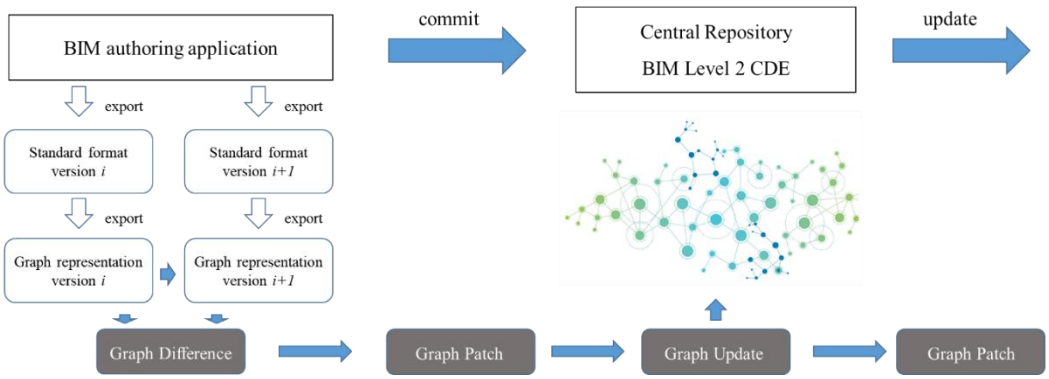

Figure 1: Basic concept of graph-based version control for distributed collaborative BIM development

The use of graph structures appears to be a promising approach. As graph-based representations reflect relationships among objects, we can apply graph theory as profound formalism to analyse a given object network's topological structure. Furthermore, modern graph database systems offer a large range of methods, which help to search, compare, and analyse subsets of the stored information. This is of special interest for the proposed approach as it introduces a large flexibility to handle various data representations and implement generic functionality that can be applied to any kind of versioned data.

3.1 Proposed framework

Due to the wide range of data specifications used in AEC projects, a central aspect of the proposed system is the definition of a meta-structure that is capable of both, reflecting the specific information stored in an instance model and mapping class definitions and relationships onto a generic graph structure. The identification of differences between two versions of a domain model is subsequently based on this graph representation populated with data by the user. The calculation results in a schema independent *DiffResult*, which defines the base for an update patch. The following paragraphs discuss the chosen graph model and present an algorithmic approach to compare two graph-based representations of a domain model.

3.2 Graph characteristics and generation

To ensure applicability for a wide range of schema specifications, a generic graph meta model is introduced. In general, a graph consists of nodes and edges. Nodes and edge can carry additional weights (i.e., attributes in the form of key-value pairs). Furthermore, each node gets one or many labels attached, which help to identify and query a specific set of nodes. Edges can be undirected or directed (Robinson, Webber and Eifrem, 2015). A graph where vertices are associated with attributes is denoted as node attributed graph or property graph. In addition, nodes can be typed leading to a typed attributed graph (Ehrig, Prange and Taentzer, 2004).

The ability to assign attributes as key value pairs to a node matches with the object-oriented paradigm of information modelling (ISO, 1999). Accordingly, we define that each node in the graph represents one class instance. All attributes of a class are attached to the node whereas associations to another class instance are modelled with an edge to represent the relationship among both class instances.

To suit the need of a schema-independent approach, a graph meta model defines a set of rules on how a given object network of an instance model is transferred in the corresponding graph. In the scope of the current paper, we define specific kinds of node labels and formalisms on how aspects of the corresponding schema specification are considered. We use the term *instance graph* to refer to specific type of graph whose specifications are provided in this section.

Node definition

Our graph meta model defines three types of nodes: *primary nodes*, *secondary nodes*, and *connection nodes*. Most schema definitions have an abstract root class that defines a *Globally unique identifier (GUID)* attribute. Due to the inheritance mechanism, all subclasses of such a root class inherit the GUID attribute as well. All other class instances, i.e. instances of classes that does not have a GUID attribute, are represented by secondary nodes in the graph. In the IFC data model for example, classes of the resource layer representing geometry, topology, material etc. do not carry a GUID. They cannot exist independently but can only exist if referenced (directly or indirectly) by one or more entities deriving from IfcRoot. The third type of nodes are denoted as connection nodes. These nodes represent the concept of objectified relationships, which is intensively used by the IFC schema specification. They provide the ability to model one-to-many relationships between class instances and assign attributes to the relationship. Similar to primary nodes, connection nodes carry a unique identifier specified by the schema specification.

Applying these mapping principles exemplary to the IFC schema, ISO 10303 is used to define the mapping of all IFC classes to the node types. ISO 10303-11 defines an entity as "*a class of information defined by common properties*" whereas an entity instance is classified by *"a named unit or data which represents a unit of information within the class defined by an entity"* (ISO, 2004). All IFC classes are either derived from IfcRoot (i.e., have a GUID) or are contained in the resource layer. All classes listed in the resource layer are reflected as secondary nodes. Subtypes of *IfcRelationships* are mapped to connection nodes.

The notion of primary, secondary and connection node will be used to define the equality of two instance graphs and helps to find an efficient implementation of the difference calculation. The detected differences in turn can be interpreted as applied modifications to the object network.

Edge definition

An edge connects two nodes of a graph. Edges can carry an edge weight, which appears as a set of key-value attributes. We use edges to model the associations between objects in an domain model. Each edge has an attribute *relType*, which indicates the association attribute between to class instances.

Graph implementation

Figure 2 depicts a simplified scenario of two classes described in the EXPRESS modelling language (ISO, 2004). The schema definition in the upper left corner defines two entities (i.e. classes without methods). The entity *point* has three attributes with an atomic datatype REAL. The *line* has one atomic attribute "Name" and two complex attributes, which reference the instances of a *Point* entity. A possible instantiation of the given data schema is given in the upper right corner, where one instance of the *Line* entity and two instances of the *Point* entity are filled with individual attribute values.

The mapping into the graph structure follows the rules explained above: Each class instance is represented by an individual node. All attributes are directly attached to the desired node whereas associations between two class instances are modelled as directed graph edges. Each edge carries the attribute name from the parent class, from where the association was initialized. The *Line* instance has a *StartPoint* and an *EndPoint* attribute (in UML/MOF an association to another class), which is reflected by the edges depicted in the graph structure. The class instance of *ShapeElement* is handled as a primary node as it owns a GUID attribute.

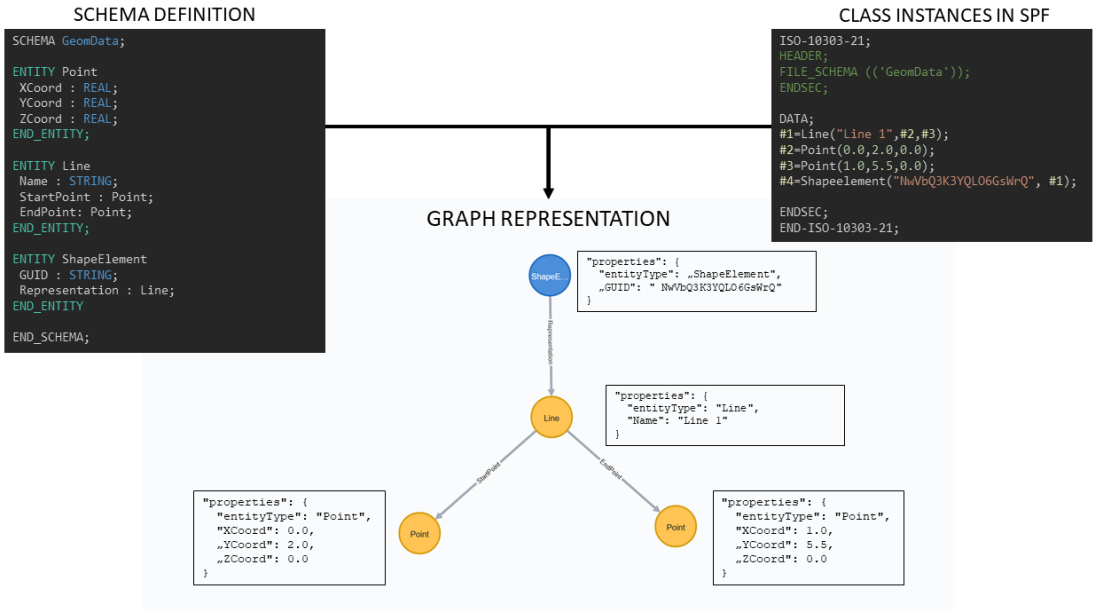

Figure 2: Correlation between schema specification, instance model, and resulting graph structure. The value stated on each edge is the value of the *relType* attribute attached to each edge.

4. Graph-based difference and update calculation

To extract the applied modification between two instance model versions, the generated graph representations of both versions are compared. Possible modifications are adding new class instances, deleting existing instances or changing associations between two instances. Also combinations of add/delete/modify can occur when comparing the object graphs.

From a mathematical point of view, the problem statement for calculating the modifications between two model versions can be defined as the following. The definition of functions follows the notation used by Kriege and Mutzel, (2012).

We denote two graphs $G_{V1} = (N_{V1}, E_{V1})$ and $G_{V2} = (N_{V2}, E_{V2})$ representing two instance model versions 1 and 2. Both are directed, labelled property graphs, where N defines the set of nodes and E the set of edges. Both, nodes, and edges, carry a weight that represents a set of key-value attributes for an individual node or edge, respectively:

$$\mathrm{W(u)}\ \forall\ \mathrm{u} \in \mathrm{N} \tag{1}$$

In addition, we define the node types using labels:

$$l_{type} \in \{\mathrm{primaryNode}, \mathrm{secondaryNode}, \mathrm{connectionNode}\} \tag{2}$$

Furthermore, an essential feature of property graphs is the flexibility to handle non-distinct node and edge sets. Accordingly, a node with a specific weight (i.e., set of attributes) can occur multiple times in the node set (Robinson, Webber and Eifrem, 2015).

The function L attaches the suitable label to a particular node.

We define a directed edge from node u to r as:

$$(u, r) \in \mathrm{E} \tag{3}$$

The aim of the update computation is to find subgraph isomorphisms between the two graphs G_{V1}, G_{V2} such as a bijective function φ can be defined:

$$\varphi\colon N_{V1} \rightarrow N_{V2} \tag{4}$$

This bijective function φ preserves adjacencies between two nodes:

$$\forall u, v \in N_{V1}\colon (u, v) \in E_{V1} \Leftrightarrow \big(\varphi(u), \varphi(v)\big) \in E_{V2} \tag{5}$$

The overall computation is spitted in two major steps. First, the structure of primary nodes and connection nodes of both graphs is compared, which results in a list of primary node tuples that are defined as equal in both model versions. Second, we compare the subgraphs of each node in the tuples from the first step and check if both subgraphs share the same information logic.

The criterion, on which two nodes or subgraphs are defined as equal, varies depending on the calculation step.

4.1 Matching primary node structures

To analyse the base skeleton of both model versions, all nodes labelled as primary nodes are retrieved from the graphs G_{V1} and G_{V2}. This operation results in two nodes sets $N_{V1,primary}$ and $N_{V2,primary}$.

Taking the weight of nodes and thereby their attributes into account, we calculate the relational intersection of both sets and declare the result as $N_{primary,unchanged}$:

$$N_{primary,unchanged} = N_{V1,primary} \cap N_{V2,primary} \tag{6}$$

All nodes in the set $N_{primary,unchanged}$ are present in both, N_{V1} and N_{V2}. Thus, no modification has been applied to the node and their attached attributes.

The relational difference between $N_{primary,unchanged}$ and $N_{V1,primary}$ results in a set of primary nodes, which are included in G_{V1}, but not in G_{V2}. Thus, the result represents a DELETE modification from version 1 to version 2:

$$N_{primary,deleted} = N_{V1,primary} - N_{primary,unchanged} \quad (7)$$

The same principle applies for nodes, which are contained in G_{V2} but not in G_{V1}, which are the result of an ADD modification from version 1 to version 2:

$$N_{primary,added} = N_{V2,primary} - N_{primary,unchanged} \quad (8)$$

Connection nodes are used to implement one-to-many relationships between primary nodes. Each connection node has directed edges, which point to primary nodes. Therefore, the aim is the analysis of the subgraph structure defined by the sets of primary nodes, connection nodes, and all corresponding edges connecting nodes of these two sets.

Therefore, we calculate the adjacency matrices of the node sets $N_{V1,primary}$, $N_{V2,primary}$, $N_{V1,con}$, $N_{V2,con}$. As we want to overcome limitations introduced by a hierarchical ordering in serialization processes, we use either the GUIDs or a calculated hashsum of each node to sort the adjacency matrix. If a relationship in both adjacency matrices is successfully identified, the corresponding *relType* attribute is checked to ensure that the detected relationship between two nodes still represents the same association.

4.2 Matching of component structures

The analysis defined in section 4.1 results in a set of unchanged primary nodes $N_{primary,unchanged}$, which is a subset of both, $N_{V1,primary}$ and $N_{V2,primary}$. As the second step, we need to analyse the subgraph structure, which is introduced by associations between a primary node and a set of secondary nodes. As depicted in Figure 2, a primary node has one or many outgoing edges pointing to secondary nodes to implement associations. Furthermore, a secondary node can have one or many outgoing edges referencing other secondary nodes. Thus, the aim of this step is the calculation of *property* modifications applied to a secondary node (i.e., adding/deleting/modifying node attributes). In addition, the network structure among secondary nodes can be modified as well, which is captured as a *structure* modification.

We define a *component* as a subgraph of the entire graph G:

$$G_{component} \subseteq G \quad (9)$$

Each component subgraph has exactly one primary node u and a set of secondary nodes $q \in N_{secondary}$, which all have a directed path P from u to a particular node q. Thus, the path P is defined by an ordered set of edges:

$$P \subseteq E = \{e_1, \dots, e_n\} \text{ connecting } u \to q \mid u \in N_{primary}, q \in N_{secondary} \quad (10)$$

To gain knowledge of structure and property modifications, the calculation is divided in several sub-steps.

First, all edges K_{V1}, K_{V2} are queried from G_{V1} and G_{V2}:

$$K_{V1} = \{(u, C(u))\} \mid (u, C(u)) \in E_{V1} \quad (11)$$

$$K_{V2} = \{(v, C(v))\} \mid (v, C(v)) \in E_{V2} \quad (12)$$

We define two edges $k_{V1} \in K_{V1}$ and $k_{V2} \in K_{V2}$ as equivalent if both carry the same value in their *relType* attribute, thus, implementing the same association between two class instances (nodes):

$$b = \begin{cases} true, \ if \ (u, C(u))_{relType} == (v, C(v))_{relType} \\ false, otherwise \end{cases} \quad (13)$$

Next, we take the nodes $q = C(u)$ and $r = C(v)$, which two equivalent edges k_{V1} and k_{V2} point towards (i.e., implement the same association), and compare the node attributes of node u against the node attributes of v. The attribute comparison detects possible property modifications and, thus, finds recently added, deleted, or modified attributes.

If an edge $k_{V1} \in K_{V1}$ exists, which has no counterpart in K_{V2}, we detect a structure modification as k_{V1} got deleted from version 1 to version 2. If an edge k_{V2} is only present in K_{V2}, but no correlation in K_{V1} can be found, we handle a structure modification of type *add* from version 1 to version 2.

To analyse the entire component (i.e., subgraph) structure, we recursively repeat the process denoted in eq. 16 and 17 with the current nodes $C(u)$ and $C(v)$. The recursion limit is reached if a node has no outgoing edges anymore (leaf node).

5. Result and discussion

The presented approach overcomes the limitations of pure file-based versioning systems by introducing a graph-based representation of instance models and their comparison by computing the graph difference. The proposed criteria on which two nodes are defined to be equal enables the user to detect not only structural modifications such as added or deleted nodes but also to find modified attribute values.

The proposed concept was tested with IFC-based instance models from various BIM authoring tools and has shown promising results. Particularly challenging, however, are complex scenarios where the attribute value is composed of nested lists. A critical example is the IFC entity *CartesianPointList3D* (ISO, 2019):

```
ENTITY IfcCartesianPointList3D
 SUBTYPE OF (IfcCartesianPointList);
  CoordList : LIST [1:?] OF LIST [3:3] OF IfcLengthMeasure;
END_ENTITY;
```

Similar discussions have appeared in the scope of ontology representations (Pauwels *et al.*, 2015).

Despite these issues, the tested prototype exposes sufficient results, which provide the base for a patch-based collaboration system.

6. Conclusion and outlook

As wide range of software applications already provide export and import interfaces to exchange BIM models on a file basis, improved techniques are required to version models on a component basis. The proposed system overcomes current limitations of a file-based data exchange by abstracting the given information in a domain model into a graph-based representation. By analyzing the topological structure and the attribute data, we can identify the applied modification between two versions by means of graph analysis. On this basis, we will develop a patch-based update system, which is capable to replace the file-based data exchange and overcomes its limitations.

As a subsequent step, the formulation of update patches will the next essential development including conflict management concepts. Furthermore, we envision not only an update transfer within a single data specification but also hope to integrate update patches between several schema specifications. Such scenarios must be handled to ensure the consistency of the resulting overall project information.

7. References

BIM Vision (2021) ‘Module: Compare’.

Blischak, J. D., Davenport, E. R. and Wilson, G. (2016) ‘A Quick Introduction to Version Control with Git and GitHub’, *PLoS Computational Biology*, 12(1), pp. 1–18. doi: 10.1371/journal.pcbi.1004668.

DIN (2019) *DIN SPEC 91391-2: Gemeinsame Datenumgebungen (CDE) für BIM-Projekte – Funktionen und offener Datenaustausch zwischen Plattformen unterschiedlicher Hersteller – Teil 2: Offener Datenaustausch mit Gemeinsamen Datenumgebungen Common*. Deutschland.

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

Helms, B. and Shea, K. (2012) ‘Computational synthesis of product architectures based on object-oriented graph grammars’, *Journal of Mechanical Design, Transactions of the ASME*, 134(2). doi: 10.1115/1.4005592.

Hidders, J. (2001) *A Graph-based Update Language for Object-Oriented Data Models*. University Press Facilities, Eindhoven, the Netherlands. doi: 10.6100/IR551259.

Ismail, A., Strug, B. and Ślusarczyk, G. (2018) ‘Building Knowledge Extraction from BIM/IFC Data for Analysis in Graph Databases’, in Rutkowski, L. et al. (eds) *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Cham: Springer International Publishing, pp. 652–664. doi: 10.1007/978-3-319-91262-2_57.

ISO (1999) *ISO/IEC 2382-15*.

ISO (2004) *ISO 10303-11:2004: Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure (ISO 10303-11:1994)*. Available at: https://www.iso.org/standard/38047.html.

ISO (2019) *DIN EN ISO 16739-1: Industry Foundation Classes (IFC) für den Datenaustausch in der Bauwirtschaft und im Anlagenmanagement – Teil 1: Datenschema (ISO 16739-1:2018)*.

Kriege, N. and Mutzel, P. (2012) ‘Subgraph matching kernels for attributed graphs’, *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2, pp. 1015–1022.

Object Management Group (2019) ‘OMG Meta Object Facility (MOF) Core Specification’, *Https://Www.Omg.Org/*. Available at: https://www.omg.org/spec/MOF/About-MOF/.

Pauwels, P. *et al.* (2015) ‘Coping with lists in the ifcOWL ontology’, *EG-ICE 2015 - 22nd Workshop of the European Group of Intelligent Computing in Engineering*.

Preidel, C. *et al.* (2018) ‘Common Data Environment’, in *Building Information Modeling*. Cham: Springer International Publishing, pp. 279–291. doi: 10.1007/978-3-319-92862-3_15.

Robinson, I., Webber, J. and Eifrem, E. (2015) *Graph Databases, Joe Celko’s Complete Guide to NoSQL*. Edited by M. Beaugureau. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. doi: 10.1016/b978-0-12-407192-6.00003-0.

Shi, X. *et al.* (2018) ‘IFCdiff: A content-based automatic comparison approach for IFC files’, *Automation in Construction*, 86(June 2016), pp. 53–68. doi: 10.1016/j.autcon.2017.10.013.

Tauscher, E., Bargstädt, H.-J. and Smarsly, K. (2016) ‘Generic BIM queries based on the IFC object model using graph theory’, in *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*. Osaka.

Turk, Ž. (2001) ‘Phenomenologial foundations of conceptual product modelling in architecture, engineering and construction’, *Artificial Intelligence in Engineering*, 15(2), pp. 83–92. doi: 10.1016/S0954-1810(01)00008-5.