Technische Universität München
TUM School of Engineering and Design

Inkrementelle Versionskontrolle verteilt vorliegender Objektmodelle im Bauwesen

Sebastian Josef Esser

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Frank Petzold

Prüfende der Dissertation:

1. Prof. Dr.-Ing. André Borrmann
2. Prof. Dr.-Ing. Christian Koch
3. Prof. Dr. Jakob Beetz

Die Dissertation wurde am 20.03.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 13.05.2024 angenommen.

Vorwort

Die vorliegende Dissertation ist Rahmen meiner Tätigkeit am Lehrstuhl für Computergestützte Modellierung und Simulation der Technischen Universität München zwischen 2018 und März 2024 entstanden.

Mein großer Dank gilt Herrn Prof. Dr.-Ing. André Borrmann für sein Vertrauen und die Chance, meine Dissertation unter seiner Betreuung anzufertigen. Besonders hervorheben möchte ich dabei die stets ausgewogene Mischung zwischen fachlicher Leitung und ausreichendem Freiraum, um neue Forschungsideen entdecken und Ansätze selbst ausprobieren zu dürfen. Gleiches gilt für die großzügigen Gestaltungsmöglichkeiten, die ich in der universitären Lehre wahrnehmen durfte.

Weiter bedanke ich mich bei Herrn Prof. Dr.-Ing. Christian Koch und bei Herrn Univ.-Prof. Dr. Jakob Beetz für die Bereitschaft, den Abschluss dieser Arbeit als Gutachter zu begleiten. Es freut mich sehr, dass ich einige ihrer Vorarbeiten zur Versionsverwaltung von BIM-Modellen und deren Repräsentationen als semantische Graphen als Basis für die Konzeption meiner Arbeit aufnehmen konnte. Herrn Prof. Dr.-Ing. Frank Petzold danke ich für den Vorsitz der Prüfungskommission sowie Herrn Prof. Dr.-Ing. Stephan Freudenstein für die Rolle als Mentor. Außerordentlicher Dank gebührt darüber hinaus Herrn Prof. Dr. rer. nat. Thomas H. Kolbe, der sehr kurzfristig meiner mündlichen Doktorprüfung beiwohnte und dort als Prüfer agierte.

Großen Anteil am Gelingen dieser Arbeit hatten darüber hinaus alle Kolleginnen und Kollegen, die ich am Lehrstuhl für Computergestützte Modellierung und Simulation und in anderen Einheiten der TUM kennen lernen durfte. Herzlichen Dank für die vielen konstruktiven Diskussionen, die mich inhaltlich wie menschlich stets bereichert haben. Hervorzuheben sind im Zusammenhang mit dem Entstehen dieser Arbeit Herr Martin Slepicka und Herr Kasimir Forth, die immer ein offenes Ohr für Fragen rund um LaTeX und zu der Gestaltung aussagekräftiger Abbildungen hatten.

Schließlich gilt besonderer Dank meiner Familie und meiner Partnerin, die mich auf meinem Weg stets bestärkt haben und mir insbesondere in der Phase kurz vor der Abgabe den nötigen Freiraum für die Fertigstellung der Arbeit eingeräumt haben.

Sebastian Esser
Weilheim, Mai 2024

Abstract

Collaboration and communication are two essential aspects of Building Information Modeling (BIM). Current practices and international standards implement BIM-based collaborative approaches based on domain-specific discipline models, which are managed as loosely coupled monolithic files and are mostly coordinated manually. Accurately tracking modifications across different versions of a model is only possible with significant effort and must be repeated for each new version. This situation is mainly caused by a lack of version control methods for BIM models that manage changes based on the underlying object structures. As a result, changes between different model states often require substantial manual investigation before potential impacts on other disciplines can be evaluated. These limitations can be addressed by implementing modern approaches to digital collaboration based on object-based synchronization, often subsumed under the term BIM Level 3 maturity in the context of the increasing adoption of BIM methodology.

This work critically examines existing technological approaches to incremental version control of object-based data structures and sheds light on significant aspects of how these integrate into the context of interdisciplinary collaboration in construction engineering. Graph-based representations are used as a central medium to adequately and losslessly represent object structures contained within a BIM model and interact with model items based on their topological structure. The use of Labeled Property Graphs enables the interaction with the object networks independently of their associated data models or markup languages, providing a universal basis for subsequent considerations regarding version control and modification management. The presented methods for graph-based version control adopt established concepts and terms from software development and the products used therein. The developed approach describes incremental changes between different model versions through the application of graph transformations and exchanging them as version increments among project stakeholders. Furthermore, formal descriptions of version increments offer new opportunities for concurrently managing diverging model versions and consolidating different states back into unified models. In addition to these considerations, approaches to future collaboration platforms and the representation of interdisciplinary dependencies are discussed. By applying incremental methods, modified objects and the changed information are directly accessible, opening up new opportunities in the context of collaborative work during the design and construction phases of projects targeting the built environment.

Zusammenfassung

Zusammenarbeit und Kommunikation sind zwei wesentliche Aspekte des Building Information Modeling (BIM). Aktuelle Praktiken und internationale Standards implementieren BIM-basierte kollaborative Ansätze auf der Grundlage domänenspezifischer Disziplinmodelle, die als lose gekoppelte monolithische Dateien verwaltet und größtenteils manuell koordiniert werden. Eine präzise Verfolgung von Modifikationen über verschiedene Versionen eines Modells hinweg ist nur mit erheblichem Aufwand möglich und muss für jede neue Version wiederholt werden. Diese Situation wird hauptsächlich durch das Fehlen von Methoden zur Versionskontrolle von BIM-Modellen verursacht, die Änderungen auf der Grundlage der zugrunde liegenden Objektstrukturen verwalten. Als Ergebnis erfordern Änderungen zwischen verschiedenen Modellzuständen oft umfangreiche manuelle Untersuchungen, bevor potenzielle Auswirkungen auf andere Disziplinen bewertet werden können. Diese Einschränkungen können durch die Implementierung moderner Ansätze zur digitalen Zusammenarbeit auf der Grundlage objektbasierter Synchronisation angegangen werden, die häufig unter dem Begriff BIM Level 3 Reife im Zusammenhang mit der zunehmenden Verbreitung der BIM-Methodik subsumiert werden.

Diese Arbeit untersucht kritisch bestehende technologische Ansätze zur inkrementellen Versionskontrolle objektbasierter Datenstrukturen und beleuchtet wesentliche Aspekte, wie diese in den Kontext interdisziplinärer Zusammenarbeit im Bauwesen integriert werden. Graphbasierte Darstellungen werden als zentrales Medium verwendet, um Objektstrukturen innerhalb eines BIM-Modells verlustfrei darzustellen und mit einzelnen Modellkomponenten auf der Grundlage ihrer topologischen Struktur zu interagieren. Die Verwendung von Labeled Property Graphs ermöglicht die Interaktion mit den Objektnetzwerken unabhängig von ihren zugehörigen Datenmodellen oder Auszeichnungssprachen und bietet eine universelle Grundlage für anschließende Überlegungen zur Versionskontrolle und Änderungsverwaltung. Die präsentierten Methoden zur graphbasierten Versionskontrolle von BIM-Modellen orientieren sich an etablierten Konzepten und Begriffen aus der Softwareentwicklung und den dort verwendeten Produkten.
Der entwickelte Ansatz beschreibt inkrementelle Änderungen zwischen verschiedenen Modellversionen durch die Anwendung von Graphtransformationen und den Austausch dieser als Versionsinkremente zwischen Projektbeteiligten. Darüber hinaus bietet die formale Beschreibung von Versionsinkrementen neue Möglichkeiten, divergierende Modellversionen parallel zu verwalten und verschiedene Zustände wieder in konsolidierte Modelle zusammenzuführen. Neben diesen Überlegungen werden Ansätze für zukünftige Kollaborationsplattformen und die Verwaltung interdisziplinärer Abhängigkeiten diskutiert. Durch die Anwendung inkrementeller Methoden sind modifizierte Objekte und die geänderten Informationen direkt zugänglich und eröffnen neue Möglichkeiten im Kontext der kollaborativen Zusammenarbeit während der Planungs- und Bauphasen von Projekten im Kontext der gebauten Umwelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

ACID Atomicity, Consistency, Isolation, Durability
ANSI American National-Standards Institute
API Application Programming Interface
ASCII American Standard Code for Information Interchange

BCF BIM Collaboration Format
BIM Building Information Modeling
bSI BuildingSMART International

CAD Computer Aided Design
CDE Common Data Environment

DAG Directed Acyclic Graph
DOM Document Object Model
DPO Double Push Out
DTD Document Type Declaration

GIS Geographisches Informationssystem
GML Generalized Markup Language
GQL Graph Query Language
GUID Globally Unique Identifier

HTML HyperText Markup Language

IEC International Electrotechnical Commission
IETF Internet Engineering Task Force
IFC Industry Foundation Classes
IRI Internationalized Resource Identifier
ISO Internationale Organisation für Standardisierung

JSON JavaScript Object Notation

KI Künstliche Intelligenz
KML Keyhole Markup Language

LPG Labeled Property Graph
LST Leit- und Sicherungstechnik

MCS Maximum Common Subgraph

MOF Meta Object Facility

NAS Normbasierte Austauschschnittstelle

OGC Open Geospatial Consortium
OWL Web Ontology Language

RDF Resource Description Framework
RFC Request for Comments

SAX Simple API for XML
SDAI Standard Data Access Interface
SGML Standard Generalized Markup Language
SPF STEP Physical File
SPO Single Push Out
SQL Structured Query Language
STEP Standard for the Exchange of Product Model Data

UML Unified Modeling Language
URI Uniform Resource Identifier
URL Uniform Resource Locator
UTF Unicode Transformation Format
UTF-16 Unicode Transformation Format - 16 bits
UTF-8 Unicode Transformation Format - 8 bits

W3C World Wide Web Consortium
WKT Well-Known-Text

XML Extensible Markup Language
XSD XML Schema Definition

Kapitel 1

Einführung

Mit der stetig fortschreitenden Digitalisierung stehen zahlreiche neue Technologien zur Verfügung, die verschiedenste Aufgaben und Prozesse im industriellen und gesellschaftlichen Bereich vereinfachen. Der Zugang zu Informationen wird durch die Vernetzung verschiedenster Akteure stetig einfacher, wobei gleichzeitig judikative und disziplinarische Paradigmen nicht oder nur verzögert mit den Möglichkeiten der Kommunikation wachsen. Damit befinden wir uns gegenwärtig in einer stetigen Abwägung zwischen explorativem Testen neuartiger Ansätze, die Innovation und Kreativität fördern, und notwendiger Reglementierung. Gleichzeitig ist zu beobachten, dass die Verständigung auf gemeinsame Ansprüche und Standards teils nicht nur durch die technologischen Errungenschaften, sondern häufig durch die Existenz historisch entstandener Verhaltensweisen erschwert wird. Die Ambivalenz von Innovationstrieb und Berücksichtigung gewachsener Strukturen ist insbesondere in der Bauindustrie zu beobachten, die sich nach wie vor durch ihre zahlreichen hochspezialisierten Disziplinen, einer großen Zahl an Akteuren innerhalb eines Bauprojekts und nicht zuletzt durch die kontinuierliche Herstellung von Unikaten auszeichnet. Damit erwachsen insbesondere an moderne Kommunikations- und Informationsmethoden zusätzliche Anforderungen, die ebendiese etablierten Verhaltensweisen in geeigneter Form erfassen und gleichzeitig technologischen Fortschritt in die Branche einführen. Absehbar ist, dass mit verbesserter Informationsgrundlage bessere Entscheidungen getroffen werden können.

Mit der Methode des Building Information Modelings wurden wesentliche Konzepte entwickelt, die den vernetzten Informationsaustausch über Unternehmens- und Disziplingrenzen hinweg fördern und damit Entscheidungen auf grundlegenden Informationen ermöglichen. Die Basis für solche Überlegungen bildet die Erkenntnis, dass Kommunikation vor allem dann funktioniert, wenn Sender und Empfänger die übermittelten Informationen in gleicher Weise verstehen und interpretieren. Zweifelsohne ist dies eine Herausforderung, die bei jeder Art von Kommunikation entsteht und die nicht allgemeingültig gelöst werden kann. Durch den Fortschritt in der Entwicklung offener, standardisierter Datenmodelle im Bauwesen ergibt sich aber nun die Chance, insbesondere die Kommunikation in kollaborativen und iterativen Szenarien weiter zu optimieren und so zukünftig insbesondere den Prozess zu einer gemeinsamen Entscheidungsfindung oder auch zu einer optimalen Entwurfslösung im ingenieurmäßigen Sinne nochmals deutlich zu verbessern.

1.1 Zielsetzung und Forschungsfragen

Die grundsätzliche Zielsetzung der vorliegenden Dissertation besteht in der Konzeption eines Systems, welches heutige Techniken der BIM-basierten, interdisziplinären Zusammenarbeit aufgreift und diese um Methoden der inkrementellen Versionskontrolle verteilt vorliegender Disziplinmodelle erweitert. Angenommen wird, dass der Austausch planerischer Informationen in Form sogenannter BIM-Modelle erfolgt, die bisher als monolithische Dateien zwischen einzelnen Fachapplikationen oder mit anderen Projektbeteiligten ausgetauscht werden. Um die eingesetzten Softwareprodukte in jeder an einem Projekt involvierten Disziplin möglichst flexibel zu halten, hat sich in den letzten Jahren der zunehmende Einsatz von herstellerneutralen Produktdatenmodellen durchgesetzt. Daher werden diese digitalen Repräsentationen mit besonderem Augenmerk behandelt und proprietäre Lösungen einzelner Softwarehersteller nur am Rande betrachtet.

Wenngleich für die verteilte Bearbeitung von BIM-Modellen heute bereits verschiedene Softwareapplikationen existieren, decken diese bisher nur bedingt die Tatsache ab, dass der Entwurfsprozess von Bauwerken in aller Regel einen iterativen Charakter aufweist und somit häufig Änderungen an bereits geteilten Informationen notwendig werden. Die Kommunikation dieser Änderungen wird bisher durch die schlichte Erstellung einer neuen Datei gelöst, die nochmalig mit allen Akteuren eines Projektes geteilt wird. Obwohl dieser Kommunikationsprozess zumindest schon einen theoretischen Zugang zu den veränderten Informationen bietet, so bedarf es auf empfangender Seite eines umfangreichen Abgleiches zwischen den verschiedenen Versionen eines Datensatzes, um die durchgeführten Änderungen und damit die möglichen Auswirkungen auf die eigenen Entwurfsentscheidungen zu bewerten. Offensichtlich wird damit, dass die Kommunikationsqualität deutlich steigen wird, wenn der repetitive Abgleich zwischen zwei Versionen eines Datensatzes bereits auf sendender Seite umgesetzt wird und so allen Empfängern direkter Zugang zu den durchgeführten Änderungen ermöglicht wird.

Konkret untersucht die vorliegende Dissertation folgende Forschungsfragen:

Forschungsfrage 1

Wie können in der Softwareentwicklung etablierte Prinzipien der optimistischen Versionskontrolle auf planerische Prozesse im Bauwesen übertragen werden?

Forschungsfrage 2

Inwiefern eignen sich Konzepte der Graphtheorie zur Beschreibung der in BIM-Modellen vorliegenden hochvernetzten Informationen und einer verlustfreien Übertragung von Modell-Veränderungen ?

Forschungsfrage 3

Wie können inkrementelle Änderungen zwischen Modellversionen abstrahiert und unter Berücksichtigung bestehender Standards interdisziplinär ausgetauscht werden?

Forschungsfrage 4

Worin bestehen Limitationen eines objektbasierten Versionskontrollsystems für BIM-Modelle?

1.2 Analogien zu textbasierten Versionierungssystemen

Die aufgeworfenen Fragestellungen weisen umfangreiche Parallelen zu bestehenden Versionskontrollsystemen auf, die bereits seit Längerem im Bereich der Softwareentwicklung zum Einsatz kommen. Wenngleich im weiteren Verlauf verschiedene Analogien zu Begriffen und Methoden textbasierter Versionskontrollsysteme herangezogen werden, benötigt die umfassende Beschreibung inkrementeller Änderungen von BIM-Modellen zusätzliche Konzeptionen, die in bestehenden Systemen bisher nicht integriert sind. Die zugrundeliegenden Limitationen werden ausführlich in Kapitel 2 beleuchtet. Dennoch werden aus Gründen der Nachvollziehbarkeit bestimmte Begrifflichkeiten aus bestehenden Versionskontrollsystemen für Quellcode in den hier vorgestellten Ansatz übernommen und entsprechend beschrieben. Gemeinsamkeiten und wesentliche Unterschiede werden entsprechend hervorgehoben und bewertet.

Als Referenz für Quellcode-Versionskontrollsysteme wird dabei auf *Git*[1] und die zugehörigen Serverdienste *Github*[2] bzw. die quellcodeoffene Alternative *GitLab*[3] Bezug genommen. Diese Produkte weisen wesentliche State-of-the-Art Funktionalitäten auf, die stellvertretend auch in anderen Plattformen mit ähnlicher Zielsetzung zu finden sind.

1.3 Aufbau der Arbeit

Kapitel 2 erläutert wesentliche Merkmale der BIM-Methodik sowie die heutigen Möglichkeiten kollaborativen Arbeitens und stellt diese den bestehenden Technologien zur Versionskontrolle gegenüber. In Kapitel 3 werden wichtige Grundlagen der Graphtheorie und der Graphtransformation eingeführt und Anwendung graphbasierter Repräsentationen in der Modellierung und Simulation von Problemen des Bauwesens vorgestellt. Kapitel 4 erörtert getroffene Abwägungen und die konzeptionellen Ansätze zur Erfassung von Modifikationen zwischen zwei Modellversionen, deren formaler Beschreibung als Graphtransformation sowie deren Anwendung auf ein überholtes Modell. Kapitel 5 führt diese Grundlagen fort und stellt die Umsetzung eines vollwertigen Versionskontrollsystems für BIM-Modelle vor. Dabei wird einerseits auf die Verwaltung chronologisch sortierter Versionsinkremente und andererseits der Umgang mit divergierenden Modellzuständen behandelt. Kapitel 6 stellt die Verwendung des entwickelten Systems anhand eines beispielhaften Workflows zwischen einem Architekten und einem Tragwerksplaner dar und erläutert im Detail die Zusammensetzung der Inkremente, die im Verlauf des

[1] https://git-scm.com/ (Letzter Zugriff am 02.03.2024)
[2] https://github.com/ (Letzter Zugriff am 02.03.2024)
[3] https://about.gitlab.com/de-de/ (Letzter Zugriff am 02.03.2024)

beschriebenen Kollaborationsprozesses ausgetauscht werden. Kapitel 7 erörtert, inwiefern die gestellten Forschungsfragen beantwortet wurden. Ebenso werden verbleibende Limitation der gewählten Ansätze diskutiert und mögliche Perspektiven für zukünftige Forschungsarbeiten skizziert. Die Arbeit schließt mit einer Zusammenfassung in Kapitel 8.

1.4 Publikationen im Rahmen dieser Arbeit

Teile der in der vorliegenden Dissertation erläuterten Aspekte wurden bereits in Konferenzbeiträgen und Fachzeitschriften veröffentlicht und werden an geeigneten Stellen aufgegriffen.

Esser, S. & Borrmann, A. (2019). Integrating Railway Subdomain-Specific Data Standards into a common IFC-based Data Model. *26th International Workshop on Intelligent Computing in Engineering*, Leuven, Belgien

Esser, S. & Borrmann, A. (2021). A system architecture ensuring consistency among distributed, heterogeneous information models for civil infrastructure projects. *Proc. of the 13th European Conference on Product & Process Modelling*. Moskau, Russland

Esser, S., Vilgertshofer, S. & Borrmann, A. (2021). Graph-based version control for asynchronous BIM level 3 collaboration. *EG-ICE 2021 Workshop on Intelligent Computing in Engineering*, pp. 98-–107, Berlin, Deutschland
https://doi.org/10.14279/depositonce-12021

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

Esser, S., Abualdenien, J., Vilgertshofer, S. & Borrmann, A. (2022). Requirements for event-driven architectures in open BIM collaboration. *Proceedings of the 29th EG-ICE International Workshop on Intelligent Computing in Engineering*, pp. 45-–53, Aarhus, Dänemark
https://doi.org/10.7146/aul.455.c195

Esser, S., Vilgertshofer, S., & Borrmann, A. (2023). A reference framework enabling temporal scalability of object-based synchronization in BIM level 3 systems. *2023 European Conference on Computing in Construction*, *177*, Heraklion, Griechenland
https://doi.org/10.35490/EC3.2023.177

Esser, S., Vilgertshofer, S. & Borrmann, A. (2023). Version control for asynchronous BIM collaboration: Model merging through graph analysis and transformation. *Automation in Construction*, *155*, 105063,
https://doi.org/10.1016/j.autcon.2023.105063

Kapitel 2

Stand der Wissenschaft zum Informationsaustausch und der modellbasierten Zusammenarbeit im Bauwesen

Im Folgenden werden wesentliche Grundlagen kollaborativen Arbeitens im Kontext der BIM-Methodik dargestellt. Der wesentliche Fokus liegt dabei zum einen auf dem Begriff des Modells, zum anderen auf der etablierten Praxis zum Austausch solcher Modelle. Im Anschluss werden die wesentlichen Limitationen abgeleitet, die bestehende Verfahren haben, und mögliche alternative Ansätze erläutert.

2.1 Grundlagen kollaborativer Ansätze im Bauwesen

Die Vision, verschiedene Informationsquellen zu vernetzten Datenstrukturen zu kombinieren und damit komplexe Sachverhalte effizient zu durchdringen, ist Grundlage für verschiedenste moderne Informationssysteme. Mit stetig leistungsfähigeren Prozessoren und Speichermedien ist es nunmehr möglich, auch rechenintensive Prozesse nahezu vollkommen orts- und zeitunabhängig durchzuführen. Gleichzeitig ermöglicht der mittlerweile fast flächendeckende Zugang zum Internet umfangreiche Möglichkeiten zur Verbreitung von Informationen, die flexibel über verschiedene Gerätearten erfasst, verarbeitet, versendet und empfangen werden können.

Diese Beobachtungen zeigen, dass auch die Bauindustrie hierdurch einen enormen technologischen Schub erfahren hat oder derzeit erfährt. In der Tat steigt die Zahl verfügbarer Technologien für verschiedene Aufgaben in der Planung, Bauausführung oder in der Betriebsphase eines Bauwerks stetig. Im Vergleich zu anderen Industriezweigen zeigt sich allerdings eine deutlich langsamere Einführung innovativer Technologien. Als erstes und mitunter wichtigstes Hindernis sei die generelle Beschaffenheit des Bauwesens genannt. Diese Industrie gestaltet sich als sehr kleinteilig sowie vielgestaltig und zeichnet sich gleichzeitig durch mitunter enormen planerischen Komplexitäten in einzelnen Projekten aus (Luo et al., 2017). Auch aus wirtschaftlicher Sicht präsentiert sich die Branche ebenfalls höchst divers. Es existieren neben großen Unternehmen mit einem vielfältigen Leistungsportfolio unzählige kleine und mittelständische Unternehmen, die sich auf einzelne Tätigkeitsfelder spezialisiert haben. Insbesondere bei Letztgenannten fällt es in der Regel schwer, große technologische Veränderungen zu vollziehen, da sie über

limitierte Ressourcen und Investitionsmöglichkeiten verfügen, deren zukünftiger Gewinn nicht explizit absehbar ist.

Die hohe Diversität der Branche führt zwar in Belangen des Marktwettbewerbs zu positiven Effekten, bedingt aber auch ein starkes Fokussieren auf individuelle Belange einzelner Akteure in einem Projekt. Das Interesse, bereits erbrachte planerische Ergebnisse in geeigneten digitalen Repräsentationen zu dokumentieren, ist damit häufig nicht in ausreichendem Umfang gegeben oder wird bisher von Auftraggebern nicht mit dem notwendigen Nachdruck eingefordert. Dabei bleibt insbesondere die für komplexe Bauwerke essentielle Vision einer optimalen gesamtheitlichen Lösung häufig hinter subjektiven Interessen und Möglichkeiten einzelner Akteure zurück. In Deutschland wird die beschriebene Problematik insbesondere bei Großprojekten ersichtlich, die teils erhebliche Projektverzögerungen und damit verbunden deutliche Kostensteigerungen erfahren haben.

Während große Unternehmen in anderen Industriesparten (bspw. Automobil- oder Luftfahrtindustrie) aufgrund ihrer Marktdominanz viel gezielter konkrete Anforderungen an den Informationsfluss zwischen Projektbeteiligten durchsetzen können, sind solche Initiativen im Bauwesen nur in kleinen Dimensionen zu beobachten. Umso mehr sind daher öffentliche Auftraggeber gefordert, die Verbesserung des Informationsaustauschs zwischen einzelnen Unternehmen, Fachdisziplinen oder Planern proaktiv voranzutreiben. In Deutschland muss gemäß Vergabe- und Vertragsordnung für Bauleistungen bis in späte Planungsphasen eine Produkt- und Herstellerneutralität gewährleistet werden. Diese besagt, dass Planungs- und Ausschreibungsdokumente keine herstellerspezifischen Aspekte beinhalten dürfen, solange kein entsprechender Zuschlag an den entsprechenden Hersteller ausgesprochen wurde. Dieser Grundsatz betrifft auch den Einsatz digitaler Planungswerkzeuge. Daher ist es insbesondere für öffentliche Vergaben essenziell, offene und produktneutrale Repräsentationen von Auftragnehmern einzufordern, sodass diese die Freiheit in der Wahl geeigneter Softwareprodukte besitzen.

Zuletzt sei in diesem Kontext noch das historisch gewachsene Vertrags- und Vergaberecht genannt, welches bei öffentlichen Bauvorhaben erneut den Marktwettbewerb fördert, aber gleichzeitig kaum Anreize zur disziplin- oder leistungsphasenübergreifenden Bereitstellung bereits getätigter Planung setzt. Es ist daher von zentraler Bedeutung, dass Auftraggeber (sowohl im öffentlichen als auch privaten Sektor) die Anwendung passender Technologien einfordern und deren Anwendung aktiv überwachen (Borrmann et al., 2021a). Die Methodik des Building Information Modelings zielt nun darauf ab, die zuvor adressierten Medienbrüche an Übergabepunkten zwischen verschiedenen Projektbeteiligten adäquat zu adressieren und maßgeblich zu verbessern.

2.2 Die Methode Building Information Modeling und ihre Reifegrade

Der Begriff des Building Information Modelings wurde erstmals von van Nederveen und Tolman (1992) verwendet, wobei die ersten Überlegungen zur digitalen Abbildung von

Gebäudemodellen (damals noch unter dem Begriff *Building Description System*) bereits Mitte der 1970er Jahren dokumentiert wurden (Eastman et al., 1974). Im deutschsprachigen Raum ist vor allem das Buch von Borrmann et al. (2021a) zahlreich zitiert worden, da es ein umfassendes Kompendium der BIM-Methodik und kürzlichen Entwicklungen in Wissenschaft und Praxis bietet.

Die konkreten Definitionen der Abkürzung BIM variieren über verschiedene Autoren und Forschungsgruppen. So wird mit BIM die Methodik definiert, die die Verwendung einer digitalen Repräsentation über den gesamten Lebenszyklus eines Bauwerks vorsieht (Borrmann et al., 2021a; Eastman et al., 2011; Verein Deutscher Ingenieure, 2020). Andere Autoren beziehen sich darüber hinaus bei der Verwendung von BIM auf die konkrete digitale Repräsentation, das Ergebnisse der planerischen Vorgänge beinhaltet.

Im weiteren Verlauf der Arbeit wird die Abkürzung BIM aber als Akronym für die Methodik und die Arbeitsweise herangezogen und der Begriff des BIM-Modells für die konkrete Instanz einer digitalen Bauwerksrepräsentation verwendet. Diese zeichnen sich nach den Definitionen von Eastman et al. (2011) durch die folgenden Charakteristika aus:

- Digitale Repräsentation
- Räumliche, dreidimensionale Darstellung
- Messbarkeit und Quantifizierbarkeit von Objekten und Maßeinheiten
- Umfassende Bereitstellung von Informationen zum Entwurf, Performance, Baubarkeit inklusive finanzieller und herstellungsrelevanter Aspekte
- Durchgängiger Zugriff für alle an einem Projekt beteiligten Parteien
- Dauerhafte Repräsentation des Bauwerks, welches für alle Lebensphasen herangezogen werden kann

Mit dem in Abb. 2.1 veranschaulichten BIM-Reifegradmodell wurde in Großbritannien in den 2010er Jahren eine anschauliche Darstellung erarbeitet, die die Transformation von konventioneller, skizzenbasierter Planung hin zu informationsreichen Produktmodellen veranschaulicht (British Standards Institution, 2013). Ähnliche Definitionen wurden vorher bereits von Succar (2010) aufgeführt. Deren Ausführungen lassen sich aus technologischer Sicht auch auf andere Länder übertragen, wenngleich die Zeitpunkte schwanken, an denen öffentliche und private Akteure Leistungen in entsprechenden digitalen Formen einfordern.

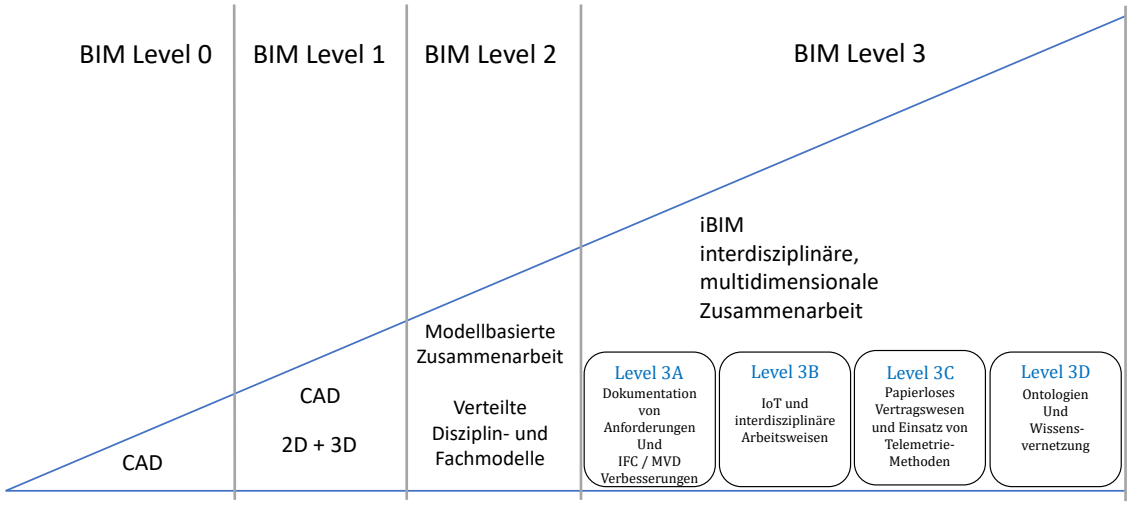


Abbildung 2.1: Die BIM Reifegrade nach Bew und Richards erweitert um die erwarteten Schritte für die BIM Level 3 Implementierung (angelehnt an Bew und Richards (2008) und Digital Built Britain (2015))

BIM Level 0 bezeichnet den konventionellen Ansatz, in dem planerische Informationen zu definierten Übergabezeitpunkten in der Regel durch das Plotten zweidimensionaler Pläne und Skizzen übermittelt werden. Durch die Verwendung unterschiedlicher, genormter Schrift- und Schraffurstile werden semantische Informationen einzelner Bauteile gekennzeichnet (Deutsches Institut für Normung e.V., 2018). Begleitend zu Planunterlagen existieren zumeist weitere Dokumente, die zusätzliche Informationen zu den Skizzen enthalten. Beispiele sind hierfür Erläuterungsberichte, Leistungsbeschreibungen, Terminpläne oder sonstige schriftliche und tabellarische Ausführungen, die Entwurfsentscheidungen und Berechnungen dokumentieren.

Durch die Fortschritte in der Leistungsfähigkeit stationärer und mobiler Computer war in der Baubranche ein zunehmendes Interesse an der dreidimensionalen geometrischen Modellierung erkennbar, die zu diesem Zeitpunkt in anderen Industriezweigen bereits umfangreiche Anwendung erfuhr. Der Einsatz dreidimensionaler geometrischer Darstellungen für planerische Aufgaben markiert den Übergang in den BIM Level 1 Reifegrad. Auch wenn dieser Fortschritt neue Möglichkeiten insbesondere zur frühen Visualisierung komplexer geometrischer Situationen (wie beispielsweise Anschlussdetails oder Knotenpunkte) ermöglicht, wurde im Sinne eines durchgängigen Informationsflusses noch kein essenzieller Fortschritt erzielt. Erst mit dem Übergang in die sogenannte Level 2 Reife wurden Ansätze aufgegriffen, die die Vision durchgängiger Informationsaustauschprozesse erahnen lassen. Kern der in dieser Entwicklungsstufe aggregierten Fortschritte ist die Umsetzung interdisziplinärer Zusammenarbeit durch Zuhilfenahme von Disziplin- und Fachmodellen und die Abwicklung von Anwendungsfällen in Form koordinierter Gesamtmodelle. Das Gesamtmodell setzt sich hierbei durch die (lose) Kopplung der einzelnen Disziplinmodelle zusammen, wobei mit jedem Disziplinmodell allerdings als eigenständige

Datei beziehungsweise Container interagiert wird (CEN, 2018). Darauf aufbauend wurde der BIM Level 3 Reifegrad dahingehend definiert, dass die Daten einzelner Disziplinen noch enger verwoben werden und zusätzliche Abhängigkeiten zwischen Objekten einzelner Disziplinen in das Gesamtmodell modelliert werden. Andere Veröffentlichungen treffen darüber hinaus die Annahme, dass die Erstellung von Disziplinmodellen mittelfristig durch das gemeinsame Bearbeiten zentral vorgehaltener Gesamtmodelle umgesetzt wird (Succar, 2010).

Im weiteren Verlauf der Arbeit werden die Ausführungen und Fragestellungen auf die Reifegrade 2 und 3 eingegrenzt. Die Prinzipien der föderierten Zusammenarbeit in Teilmodellen, die zugehörigen Standards und verbundene Limitationen werden später in Abschnitt 2.4 ausführlich dargelegt. Zuvor erläutern die nächsten Abschnitte wesentliche Grundlagen des modellbasierten Informationsaustausches und zugehörigen Techniken der Informationsserialisierung. Diese sind relevant, um grundlegende Strategien für die modellbasierte Zusammenarbeit im Bauwesen zu motivieren und bestehende Limitationen heutiger Systeme zu beschreiben.

2.3 Datenmodelle und Informationsaustausch

Wie in Abschnitt 2.1 bereits erläutert, ist der Wunsch nach Methoden zum hochqualitativen Datenaustausch im Bauwesen in den letzten beiden Jahrzehnten stetig gewachsen. Um den Nutzen modellbasierter Kollaboration in Planung, Entwurf, Bau und Betrieb gebauter Umwelt einordnen zu können, bedarf es vorab einer Eingrenzung des *Modell-Begriffs*, der in Abschnitt 2.3.1 erläutert und in Abschnitt 2.3.2 um Aspekte der Informationstechnologien erweitert wird. Für die computerauswertbare Übergabe von Modellen werden anschließend in Abschnitt 2.3.4 die wichtigsten Auszeichnungssprachen vorgestellt. Anschließend werden in Abschnitt 2.3.5 wichtige Datenmodelle vorgestellt, die heute für den Datenaustausch in Planung, Bau und Betrieb eingesetzt werden.

2.3.1 Allgemeiner Modellbegriff

Der Begriff des *Modells* wird in der Umgangssprache in verschiedenen Sachverhalten herangezogen. Einerseits beschreibt der Begriff ein *Abbild* als Rückblick bzw. *Vorbild* als Vorschau auf etwas, kann aber in der Kunst auch bedeuten, dass ein erzeugtes Abbild (z.B. Gemälde) nicht immer umfänglich identisch mit dem oder der ”Modell-Stehenden” sein muss (Kobler, 2010; Stachowiak, 1973).

Stachowiak (1973) definiert drei wesentliche Merkmale des allgemeinen Modellbegriffs:

1. Das **Abbildungsmerkmal**: Modelle repräsentieren stets ein Original, zu dem eine Abbildungsrelation besteht. Das Original kann entweder real oder abstrakt wahrnehmbare Realität oder ein Modell sein, wodurch verschiedene Abstufungen und Abstraktionen ermöglicht werden. Zusätzlich kann die durchzuführende Abstraktion

unter verschiedenen Merkmalen oder Zielsetzungen umgesetzt werden, sodass für ein Original mehrere Modelle mit verschiedenen Spezifika existieren können.

2. Das **Verkürzungsmerkmal**: Modelle können niemals alle, sondern nur die für einen konkreten Zweck relevanten Merkmale des Urbilds enthalten.

3. Das **pragmatische Merkmal**: Modelle sind ihren Originalen nicht automatisch eindeutig zugeordnet. Sie erfüllen folgende Funktionen, sind aber auch auf diese beschränkt:

   - Ein Modell erfüllt mit seiner repräsentativen Form immer nur Anforderungen bestimmter Modellnutzer, denen die gewählte, repräsentierende Form genügt.
   - Ein Modell gilt nur innerhalb bestimmter Zeitintervalle.
   - Ein Modell verfolgt grundsätzlich einem spezifischen Zweck und unterliegt damit den getroffenen Einschränkungen und Annahmen.

Hars (1994) beleuchtet den Begriff des Modells mit sogenannten *Systemen* und deren Beziehungen zueinander. Ein System ist demnach ein Gebilde von realen oder abstrakten Strukturen, die sich aus Elementen, Eigenschaften ebendieser und Beziehungen charakterisieren. Diese Definition ähnelt den vorangegangenen Begrifflichkeiten von Stachowiak (1973), erweitert diese aber um eine klare Betrachtung des Zwecks. Dabei soll ein Modell mögliche Zustände und Erscheinungsformen so erfassen, dass Aussagen, Ableitungen und Folgerungen in einem gegebenen Kontext informiert getroffen werden. Da wie beschrieben ein Modell nie exakt sein kann, leitet Hars (1994) den Begriff der Modellqualität ab. Nach seinen Ausführungen kann die Qualität demnach daran bemessen werden, wie gut der Zweck erfüllt wird, für den ein Modell definiert wurde. Damit wird der Modellbegriff um eine nutzungsbezogene Komponente erweitert.

Abb. 2.2 illustriert wesentlichen Merkmale des Modellbegriffs. In allen Fällen steht ein Original (in diesem Falle die Realität) im Zentrum der Betrachtung. Das Original wird dabei auch als *Domäne* bezeichnet, die den physikalischen Raum der Realität beschreibt. Je nach Zweck der Modellbildung werden in der Folge Teile der Realität ausgewählt und in das Modell abgebildet. Der Begriff der *Abbildung* bezeichnet dabei die Zuordnung eines Objekts der Realität auf einen entsprechenden Begriff in Modell. Im Rahmen einer Abbildung können einzelne Objekte kontrastiert werden, also feingranularer erfasst werden als andere. Außerdem können Fälle auftreten, bei denen Attribute einzelner Objekte im Modell nicht berücksichtigt oder zusätzlich in das Modell hinzugefügt werden. Im letztgenannten Fall können dies beispielsweise Informationen sein, die sich aus der menschlichen Wahrnehmung oder Transferwissen ergeben, aber kein direktes Gegenstück in der realen Welt besitzen. Daraus wird ersichtlich, dass es sich bei der Modellbildung nicht immer um bijektive Abbildungen handelt, sondern auch surjektive oder injektive Beschreibungen zum Einsatz kommen können. Auf diese Begriffe wird später in Abschnitt 3.1 genauer eingegangen.

Selbst innerhalb einer passend definierten Domäne kann es verschiedene Modellausprägungen geben, die unterschiedliche Ausdrucksweisen verwenden, aber den gleichen

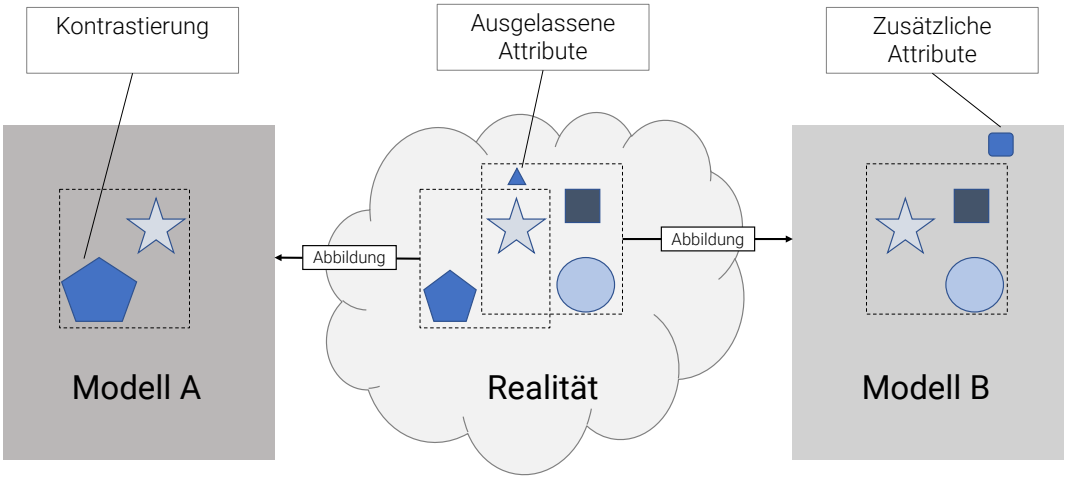
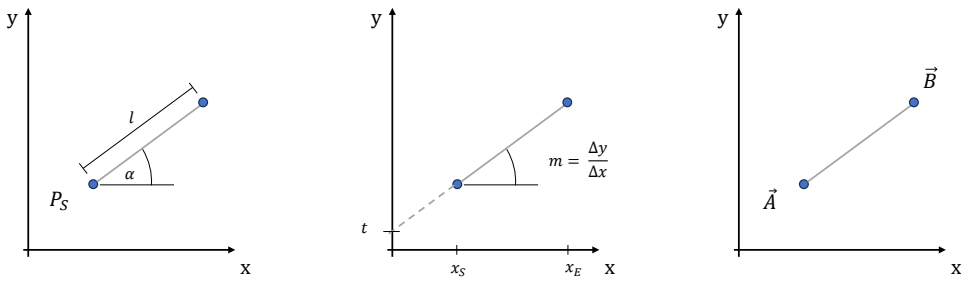


Abbildung 2.2: Zusammenhang zwischen Realität und verschiedenen Modellbildungen (inspiriert von Stachowiak (1973) und Kobler (2010))

realen Sachverhalt abbilden. Hierfür gibt es in der Mathematik zahlreiche Beispiele. Stellvertretend wird im Folgenden die mathematische Abbildung einer einfachen Strecke untersucht. Abb. 2.3 zeigt drei Möglichkeiten, die alle eine mathematisch eindeutige Beschreibung einer Strecke verwenden. Damit wird deutlich, dass es in der Modellbildung häufig mehr als eine richtige Lösung geben kann. Manche Abstraktionen mögen für bestimmte Zwecke besser geeignet sein als andere. In anderen Fällen kann darüber hinaus eine verlustfreie Übersetzung zwischen verschiedenen Sichten funktionieren. Zudem zeigt es, dass manchmal der Gültigkeitsbereich für eine Abbildung beschränkt werden muss, um die Validität der Modellierung zu bewahren. So hängt die Auswertung des Modells im zweiten und dritten Fall von dem Definitionsbereich x_S und x_E beziehungsweise von der Wahl eines geeigneten Parameters t ab, dessen Werte auf die reelen Zahlen im Bereich $[0, 1]$ beschränkt wird. Wird diese Einschränkung weggelassen, so behandelt das Modell nicht mehr das Konstrukt einer *Strecke*, sondern einer *Gerade*. Daraus lässt sich verallgemeinern, dass verschiedene Modelle an den Grenzen ihres Definitionsbereiches Charakteristiken anderer Modelle haben können, die ähnliche, artverwandte Sachverhalte beschreiben, aber gleichzeitig in ihrem eigenen Gültigkeitsbereich ebenfalls beschränkt sind.

Aus diesem einfachen Beispiel lässt sich eine Vielzahl an Herausforderungen ableiten, die in der Modellbildung und in der Verwendung des Modells Berücksichtigung finden müssen. Neben einer fundierten Kenntnis der zu beschreibenden Domäne ist es notwendig, die Grenzen der getroffenen Annahmen und Abstraktionen zu kennen. Zudem bedarf es ausreichender Expertise über die Methoden, mit der die Wirkungsweise der Modelle beschrieben werden soll. Hierfür gibt es wiederum zahlreiche Ausprägungen. Schilling (2018) zählt neben haptischen Modellen, wie sie beispielsweise nach wie vor umfangreich in der Architektur oder in der Lehre verschiedener Naturwissenschaften eingesetzt werden, auch



(P_S, α, l) | $f(x) = m \cdot x + t \mid x\ in\ [x_S, x_E]$ | $\vec{A} + t \cdot (\vec{B} - \vec{A}) \mid t\ in [0, 1]$

Abbildung 2.3: Drei Varianten zur mathematischen Beschreibung einer Strecke als Modelle

die Existenz von Objektmodellen auf. Diese eignen sich insbesondere, um Gegenstände, Verhaltensweisen, Prozesse oder deren gemeinsamer Interaktion in einem Modell zu beschreiben. Für die weiteren Untersuchungen soll der Fokus nun insbesondere auf die letztgenannte Modellart gerichtet und später auf die spezifischen Belange des Bauwesens eingegrenzt werden.

2.3.2 Objektmodelle in der Informatik

Objektmodelle können in vielfältiger Art und Weise beschrieben und dokumentiert werden. Von besonderem Interesse sind im Kontext dieser Arbeit beschreibende Methoden, die eine computerinterpretierbare Auswertung ermöglichen. Eng verknüpft ist damit der Begriff des *Datenmodells*, wofür es in der Informatik weitreichende Definitionen gibt, die je nach Anwendungskontext unterschiedliche Ausprägungen und Merkmale des Modellbegriffs priorisieren. Als verhältnismäßig allgemeingültig kann die Definition des Wirtschaftslexikons von Gabler herangezogen werden:

> *„[...] Modell der zu beschreibenden und verarbeitenden Daten eines Anwendungsbereichs (z.B. Daten des Produktionsbereichs, des Rechnungswesens oder die Gesamtheit der Unternehmensdaten) und ihrer Beziehungen zueinander.“* Gabler Wirtschaftslexikon, 2022

Ein Datenmodell erweitert demnach den allgemeinen Modellbegriff um den Aspekt der computerinterpretierbaren Beschreibung der Elemente, Eigenschaften und Beziehungen und ist dabei eng verzahnt mit der *objektorientierten Programmierung*. Datenmodelle stellen typischerweise ausschließlich die Definitionen bereit, mit welchen Attributen und Relationen eine Domäne beschrieben werden sollen. Die tatsächliche Befüllung mit konkreten Werten zur Abbildung eines realen Sachverhaltes werden *Instanzen* der im Datenmodell definierten Entitäten verwendet.

Ein Blick in die Literatur zeigt, dass frühe Vorläufer objektorientierter Paradigmen bereits in den 1960er Jahren dokumentiert wurden. Kay (1996) beschreibt, dass zu diesem Zeitpunkt noch keine Standards für Betriebssysteme oder Dateiformate existierten und verschiedene Entwicklungen je nach verfügbaren Computern und Technologien stattfanden. Dennoch

waren nach seinen Ausführungen vereinzelt Ansätze zur Trennung von Informationen und Funktionen innerhalb eines Computers erkennbar, wenngleich zu diesem Zeitpunkt der Innovationscharakter dieser Überlegung und deren Tragweite für zukünftige Systeme noch nicht absehbar waren. Dennoch reifte bereits in dieser Zeit die Überlegung, Aufgaben, die ein einziger Computer erledigen kann, in mehrere Teilaufgaben aufzuteilen, um so Skalierbarkeit und Kapselung einzelner Bestandteile eines Programms zu erreichen. Eines der ersten Systeme mit objektorientierten Strukturen war das 1963 von Ivan Sutherland entwickelte Sketchpad, welches graphische Ein- und Ausgaben geometrischer Formen auf Monitoren ermöglichte (Sutherland, 1963). Davon und von anderen Entwicklungen verschiedener Programmiersprachen inspiriert veröffentlichte 1972 Alan Kay die Sprache *Smalltalk*, die als erste objektorientierte Programmiersprache bezeichnet wird und deren Ansätze bis heute in modernen Sprachen Anwendung finden (Kay, 1996).

2.3.3 MOF-Definitionen

Das Zusammenspiel zwischen Instanzen und zugehörigem Datenmodell kann formal mit den Begriffen der Meta Object Facility (MOF) beschrieben werden. Die Motivation hinter der Meta Object Facility (MOF)-Definition erwuchs aus der Erkenntnis, dass inkompatible und in vielen Fällen proprietäre Metadaten ein wesentliches Hindernis für den Datenaustausch zwischen verschiedenen Systemen darstellen. Als Metadaten werden dabei *Daten über Daten* verstanden, die die Beschreibung der Informationsbeschaffenheit ermöglichen. MOF stellt dafür eine Umgebung für die Beschreibung und Verwaltung von Datenmodellen bereit, die jeweils mit zugehörigen Meta-Modellen beschrieben werden können. Ein Metamodell dient demnach der Spezifikation aller Begrifflichkeiten eines Modells und definiert die möglichen Konstrukte. Die einzelnen Abstraktionsstufen sind hierarchisch angeordnet und für das vierschichtige Modell in Abb. 2.4 illustriert.

Für die objektorientierte Modellbildung werden im allgemeinen Sprachgebrauch in der Regel die in MOF-M2 definierten Terme von Klassen und Attributen definiert. Der Begriff der *Klasse* beschreibt die Gruppierung ähnlicher Objekte, die gemeinsame Charakteristika besitzen, sich aber in ihrer expliziten Ausprägung unterscheiden. Neben *Klassen* existiert das Konstrukt eines *Attributs*, welches sich durch einen eindeutigen Namen und über einen Datentyp definiert, der den Speicherbedarf des Attributs angibt.

Die Konstrukte aus MOF-M2 können anschließend von beliebig vielen Modellen auf der MOF-M1 Ebene aufgegriffen werden. Typische Vertreter in dieser Ebene sind jene Strukturen, die als *Datenmodell* oder *Produktmodell* bezeichnet werden und domänenspezifische Begrifflichkeiten definieren. Die Instanzen in der Schicht M0, die von Modellen der M1-Ebene gebildet werden, stellen letztendlich die konkreten Objekte zur Laufzeit dar, die spezifische Charakteristika tragen. Im weiteren Verlauf wird die M0-Notation verwendet, um sowohl Instanzen innerhalb eines Speichermediums als auch serialisierte Formen zu adressieren. Ausführlicher wird dies in Abschnitt 2.3.5 beleuchtet.

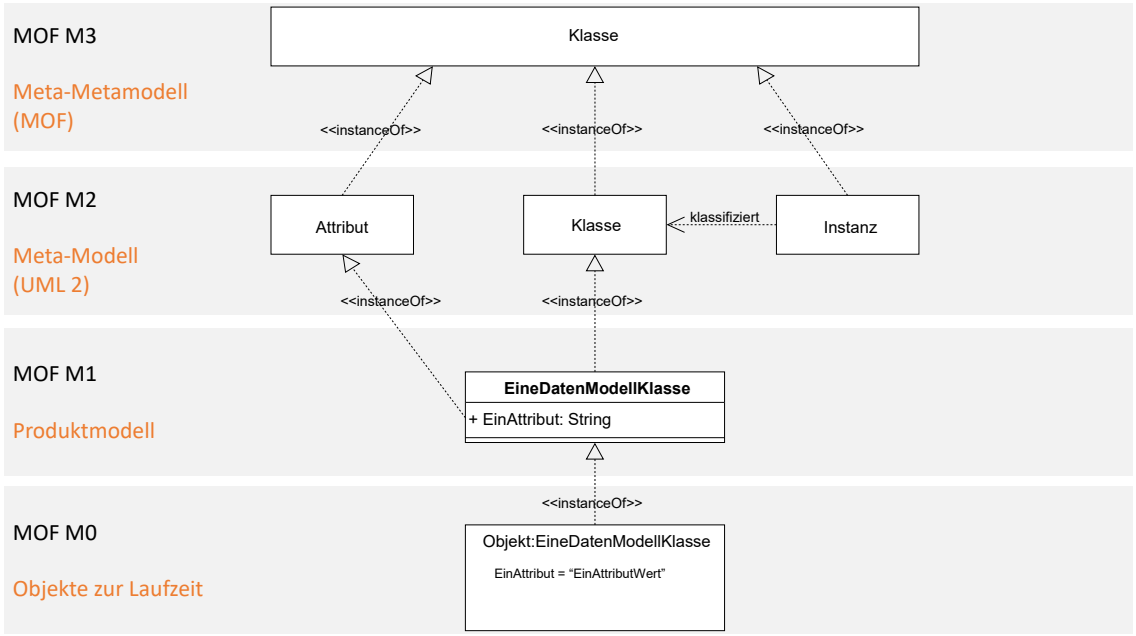


Abbildung 2.4: 4-Schichten-Metadaten-Architektur (leicht vereinfacht und angelehnt an Object Management Group (2017))

2.3.4 Zeichensatzkodierungen und Auszeichnungssprachen

Für alle Ebenen der MOF-Notation werden einheitliche Kodierungssysteme benötigt, die den jeweiligen Informationen umfassend und valide beschreiben und gleichzeitig unabhängig von dem eingesetzten Computersystem sind. Eine solches Kodierungssystem definiert dabei für ein konkretes Zeichen eine bestimmte Byte-Reihenfolge, die standardisiert von den unterstützenden Systemen gleichartig interpretiert und dargestellt werden. Vorreiter war dabei in den 1960er Jahren der amerikanische Standard-Code für den Informationsaustausch (bekannt unter der Abkürzung ASCII), der das lateinische Alphabet, diverse Interpunktionszeichen sowie nicht-druckbare Steuerzeichen wie Tabulator oder Zeilenumbrüche umfasst. Die Bitkodierung wurde zu Beginn mit 7 Bit zur Beschreibung der einzelnen Zeichen umgesetzt, sodass damit maximal 128 Zeichen kodierbar waren. Diese haben sich im wesentlichen mit den damaligen Eingabemöglichkeiten in Tastaturen von Schreibmaschinen gedeckt.

Zahlreiche natürliche Sprachen, die auf dem lateinischen Alphabet aufbauen, verwenden zusätzliche Zeichen, die beispielsweise für die Abbildung von Umlauten oder Betonungen notwendig sind. Hierfür wurde im Anschluss ein achtes Bit verwendet, welches sprachspezifische Aspekte kodieren kann. Das American National-Standards Institute (ANSI) übernahm diesen Ansatz, führt die 128 kodierten Zeichen des American Standard Code for Information Interchange (ASCII)-Standards fort und definierte weitere 63 sprachspezifische Zeichen. In Summe umfasst die gemeinhin als ANSI-Kodierungsstandard bezeichnete Zeichensammlung demnach 191 Zeichen, die im Anschluss durch die Internationale Organisation für Standardisierung und International Electrotechnical Commission als Richtlinie ISO/IEC 8859-1, auch bekannt als Latin-1, veröffentlicht wurde (ISO, 1998). Dieser genügt vor allem den Zeichenanforderungen der englischen und anderen europäischen Sprachen.

Für die Abbildung anderer Sprachen wurden weitere Standards veröffentlicht, die sich ebenfalls in die ISO 8859 Reihe einordnen (bspw. ISO 8859-5 für Türkisch oder ISO 8859-10 für Zeichen, die in südosteuropäischen Sprachen verwendet werden).

Deutlich umfangreicher als die in ASCII und ANSI definierten Zeichen ist der UniCode Zeichensatz. Diese Sammlung umfasst in der Version 15 mittlerweile knapp 150.000 Zeichen und hat zum Ziel, möglichst alle Zeichen der auf der Welt vorkommenden Sprachen zu erfassen und zu kodieren (The Unicode Consortium, 2022). Die Bitkodierung kann dabei in verschiedenen Verfahren erfolgen und unterscheidet sich in der angesetzten Bit-Basis. Am verbreitetsten sind die Varianten mit 8 und 16 Bit, die sich im Unicode Transformation Format (UTF) wiederfinden. Das Unicode Transformation Format - 8 bits (UTF-8) wird heute annähernd flächendeckend in der Definition von Web-Schnittstellen und -Protokollen eingesetzt, in Zukunft ist aber dessen Ablösung durch die UTF-16-Variante zu erwarten oder zum Teil schon in Umsetzung (z.B. bei der Einführung von HyperText Markup Language (HTML) Version 5).

Die Standardisierung der computerinterpretierbaren Zeichensätze hat ihren Ursprung in den 1960er Jahren. Dabei verfolgten die Initiativen vor allem das Ziel, Schriftstücke elektronisch zu speichern und austauschen zu können und stellten damit die ersten Schritte in der Digitalisierung von Kommunikationsgeräten dar. Schnell erwuchs damit aber der Anspruch, neben dem eigentlichen Inhalt auch Textformatierungen übermitteln zu können (Haralambous, 2007). Diese Möglichkeit wurde mit der Einführung von *Auszeichnungssprachen* geschaffen, die im Englischen auch als *Markup Languages* bezeichnet werden. Auszeichnungssprachen erweitern den Zeichensatz um zusätzliche *Auszeichnungsmerkmale*, die einerseits als verschiedene Trennzeichen zwischen einzelnen Textelementen bereitstehen (z.B. Leerzeichen und Zeilenumbruch) und andererseits spezifische Funktionen oder Formatmerkmale anzeigen (Goldfarb, 1981). Der Ursprung dieser Überlegungen entwickelte sich Ende der 1970er Jahren aus dem Wunsch, einfache Zeichenfolgen mit zusätzlichen semantischen Informationen auszustatten und beispielsweise Überschriften, Absätze oder Fußnoten als solche computerinterpretierbar darzustellen. Die Formatanweisungen wurden stets als imperative Anweisung angegeben, sodass eine *präsentationsorientierte* Beschreibung vorlag. Diese stellte sich aber als zunehmend unhandlich dar, sobald die formatierten Dokumente anderweitig interpretiert werden sollten. Die Umsetzung von funktionalen Merkmalen wie die Abgrenzung von Kapiteln war noch nicht in den imperativen Befehlen abzubilden (Ray & McIntosh, 2002).

Während zu Beginn die imperative Auszeichnung eines Textes auf eine bestimmte Anwendung zugeschnitten war, bewarb Goldfarb (1981) die Verwendung beschreibender, generischer Merkmale. Ähnlich zu den Überlegungen zur Entwicklung objektorientierter Programmiersprachen, entstand auch bei der Entwicklung der Auszeichnungssprachen der Wunsch nach Trennung von Inhalt und Formatierungsanweisungen, um verarbeitenden Systemen mehr Flexibilität bei der Darstellung und Aufbereitung der Dokumente zu ermöglichen. Der Paradigmenwechsel von imperativen zu funktionalen Auszeichnungen ermöglichte es laut Goldfarb (1981), Eigenschaften auch über verschiedene Systeme hinweg zu verarbeiten, da für das Prozessieren nunmehr drei Schritte notwendig waren:

1. Erkennung: Das System erkennt ein Auszeichnungsmerkmal.
2. Zuordnung: Die Applikation kann den Inhalt des Merkmals einer spezifischer Funktion zuweisen.
3. Prozessieren: Die passende Funktion wird ausgeführt.

Aus den erläuterten Gründen entwickelten Charles Goldfarb, Edward Mosher und Raymond Lorie die Generalized Markup Language (GML), welche 1986 durch die Internationale Organisation für Standardisierung (ISO) als Standard Generalized Markup Language (SGML) in der Norm 8879 veröffentlicht wurde (Ray & McIntosh, 2002). Damit existierte erstmals ein Mechanismus zur digitalen Beschreibung und Austausch von Objekten, der auf den international etablierten Kodierungssystemen wie ANSI und UTF aufbaute und generisch in verschiedenen Systemen, Programmiersprachen und Zielsetzungen verwendet werden konnte. Als wohl bekanntestes Derivat der SGML wird bis heute HTML für die Darstellung von Webseiten verwendet. HTML schränkt das in SGML verfügbare Vokabular deutlich ein und verwendet lediglich eine kleine Teilmenge an Merkmalen, die Darstellungsinformationen transportieren.

Extensible Markup Language (XML)

Ray und McIntosh (2002) führen an, dass SGML sehr umfangreiche Begriffe und Möglichkeiten bereitstellt, sodass sich eine flächendeckende Einführung erst einmal nicht einstellte. Die Verwendung von HTML hingegen war vor allem auf die funktionale Beschreibung von Texten und deren Darstellung ausgelegt und konnte für andere Anwendungszwecke kaum eingesetzt waren. Aus diesen Erkenntnissen (SGML zu umfangreich und HTML zu eingeschränkt) wurde ein erneuter Standardisierungsversuch unternommen, der schließlich in der Extensible Markup Language (XML) endete. Auch XML basiert wie HTML auf der SGML, verfügt aber über den großen Vorteil, dass das verfügbare Vokabular zur Beschreibung von Objekten nicht beschränkt ist.

Heute wird die Entwicklung von XML und verwandten Technologien unter dem Dach des World Wide Web Consortiums geführt. Das Konsortium ist eine internationaler Zusammenschluss verschiedener Organisationen, Unternehmen und Einzelpersonen, die gemeinsam Webstandards entwickeln und pflegen. Deren Hauptziel des W3C ist es, das World Wide Web so zu gestalten, dass es zugänglich, nutzbar und skalierbar ist. Daher spielt die Entwicklung von offenen Standards und Empfehlungen eine wesentliche Rolle. Diese sollen sicherstellen, dass das Web in einer kohärenten und interoperablen Weise funktioniert. Damit nehmen die Empfehlungen des W3C erheblichen Einfluss auf die Art und Weise, wie Webseiten und Webanwendungen erstellt und dargestellt werden. Sie sorgen dafür, dass die verschiedenen Technologien und Plattformen im Web effektiv zusammenarbeiten können, um eine konsistente Benutzererfahrung zu gewährleisten.

Darüber hinaus wurden diverse Konzepte zur Befüllung und Verwendung von XML in Request for Comments (RFC) der Internet Engineering Task Force (IETF) diskutiert. Die

IETF ist eine Organisation, die die Entwicklung von Standards und Protokollen für das Internet vorantreibt. RFCs sind formale Dokumente, die Spezifikationen, Protokolle, Verfahren, Konzepte und andere technische Informationen im Zusammenhang mit dem Internet enthalten. Ursprünglich wurden RFCs als informelle Kommunikationsmittel entwickelt, um Ideen und Vorschläge innerhalb der technischen Community auszutauschen. Im Laufe der Zeit sind sie jedoch zu offiziellen Dokumenten geworden, die Internetstandards definieren. Die IETF behandelt eine breite Palette von Themen, die von der Beschreibung von Protokollen über Best-Practice-Richtlinien bis hin zu technischen Überlegungen für zukünftige Standards reichen. Die Bezeichnung *Request for Comments* spiegelt die ursprüngliche Absicht wider, dass die Dokumente als Vorschläge zur Diskussion gestellt werden, bevor sie als Standards angenommen werden. RFCs erhalten eine eindeutige Nummerierung und sind öffentlich zugänglich. Die Nummerierung erfolgt chronologisch und startete mit der Nummer 1 im Jahr 1969. In Abgrenzung zu World Wide Web Consortium (W3C) befasst sich die IETF vorrangig mit Aspekten des Datentransfers, wohingegen W3C eher Aspekte der Nutzbarkeit und Zugänglichmachung bearbeitet.

Die Auszeichnungssprache XML findet in verschiedensten Szenarien Anwendung, die eine standardisierte Informationsformatierung benötigen. Die Kernidee hinter XML besteht in der Beschreibung von Daten anhand ihrer Struktur und ihres Zwecks. Gleichzeitig gibt XML keine Anweisungen für eine bestimmte Aufgabe oder Interpretation dieser Informationen. Die für die Darstellung zusätzlich notwendigen Informationen werden stattdessen in einem StyleSheet oder anderen begleitenden Ressourcen gespeichert. XML verwendet wie auch SGML sogenannte *Tags*, die die Inhalte eines *Elements* umschließen. *Elemente* sind die elementaren Bausteine, aus denen sich ein XML-Dokument zusammensetzt. Weiterhin können innerhalb der Element-Tags zusätzliche Attribute in Form von Name-Wert-Paaren hinterlegt werden. Alle *Elemente* innerhalb eines XML-Dokuments müssen sich innerhalb eines so genannten *RootElements* befinden und werden hierarchisch angeordnet. Infolgedessen entsteht aus der Gesamtheit aller *Elements* in einem XML-Dokument eine Baumstruktur. Es besteht aber durch Zuhilfenahme von *XLink* und *XPointer* die Möglichkeit, Querverweise in die hierarchische Struktur einzuführen und damit Verweise auf beliebigen Ebenen herzustellen [1].

Algorithmus 2.1 zeigt ein einfaches Beispiel eines XML-Dokuments, welches die in Abb. 2.5 gegebene Situation beschreibt. Das XML-Dokument enthält Informationen über die Strecken, die sich jeweils durch einen Start- und Endpunkt definieren.

[1] https://www.w3schools.com/xml/xml_xlink.asp (letzter Zugriff am 02.01.2024)

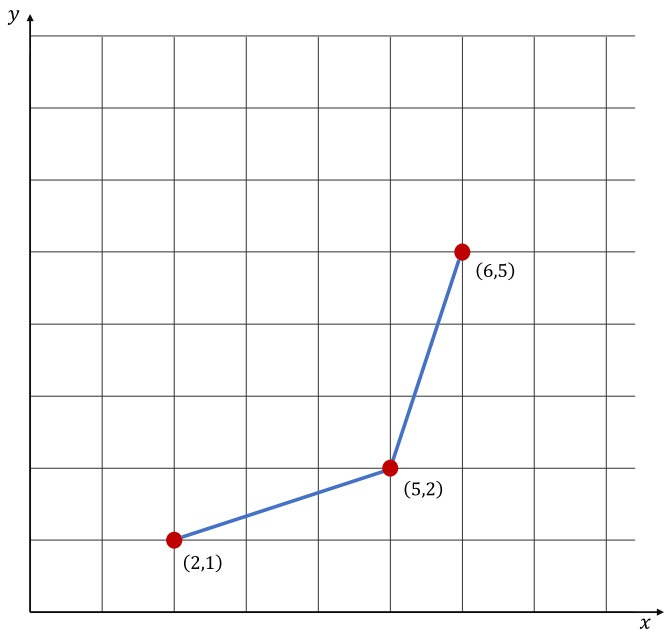


Abbildung 2.5: Geometrisches Modell, welches zwei Strecken mit ihren Start- und Endpunkten beschreibt

Algorithmus 2.1: Einfaches XML Beispiel zur Beschreibung der in Abb. 2.5 illustrierten Domäne

```
<Linienbeispiel>
    <Strecke>
        <Start>
            <Punkt XWert=2.0 YWert=1.0 />
        </Start>
        <Ende>
            <Punkt XWert=5.0 YWert=1.0 />
        </Ende>
    </Strecke>
    <Strecke>
        <Start>
            <Punkt XWert=5.0 YWert=2.0 />
        </Start>
        <Ende>
            <Punkt XWert=6.0 YWert=5.0 />
        </Ende>
    </Strecke>
</Linienbeispiel>
```

Neben der Beschreibung einer Information in der MOF-MO Schicht kann XML auch auf MOF-M1 Ebene zur Deklaration der Datenstrukturierung in Form eines Datenmodells eingesetzt werden. Hierfür wird die XML Schema Definition eingesetzt und fungiert damit als Document Type Declaration (DTD). XML Schema Definition (XSD)-Dokumente werden ebenfalls mit XML-Syntax formuliert, spezifizieren aber die Beschaffenheit der Daten und ermöglichen in bestimmten Grenzen auch die Abbildung von Regeln und einfachen Funktionen (Walmsley, 2001). XSD-Dokumenten können prinzipiell alle grundlegenden Prinzipien objektorientierter Modellierung wie Klassen, Attribute mit Datentypen, Bezie-

hungen zwischen Klassen sowie deren Kardinalitäten und in begrenztem Umfang auch Funktionen beschreiben.

Wird keine DTD verwendet, können XML-Dokumente auch einer freien Form (wie in Algorithmus 2.1 gezeigt) folgen. In diesen Fällen muss trotzdem die *Wohlgeformtheit* des Dokuments erfüllt sein. Diese besagt, dass alle geöffneten *Tags* jeweils auch ein schließendes Pendant haben müssen und sich insgesamt die besagte hierarchische, für XML-Dokumente charakteristische Struktur einstellt. Ein XML-Dokument kann per Definition jede der zuvor erläuterten Kodierungssysteme (ASCII und UTF-Varianten) verwenden und wird in Form eines *declaration*-Elements für das lesende System vorgehalten (Ray & McIntosh, 2002).

Zur Interaktion mit XML-Dokumenten existieren zwei wesentliche Ansätze, wie eine passende Application Programming Interface (API)-Schnittstelle in Programmen umgesetzt werden kann. Das Document Object Model (DOM) wandelt den gesamten Inhalt einer XML-Datei in eine objektorientierte Struktur im Speicher des prozessierenden Geräts. Davon ausgehend ist im es im Anschluss möglich, umfangreiche Traversierungsfunktionen auf der gesamten Objektstruktur zu verwenden. Die Ausführungen zu der Simple API for XML (SAX) weisen im Kontrast eine event-basierte Architektur auf und verarbeitet immer nur jene Informationen, die dem Parser zum jeweiligen Zeitpunkt zur Verfügung stehen. In der Literatur sind beide Ansätze ausführlich behandelt worden. Interessierten Lesern sei daher beispielsweise das Werk von Harold und Means (2004) empfohlen.

Standard for the Exchange of Product Model Data (STEP)

Annähernd parallel zu den Entwicklungen um SGML und XML erfolgte in den 1980er Jahren ein enormer Entwicklungsschub im Bereich des Computer Aided Design (CAD). Damit einher ging auch die Vision, Produkte und zugehörige Informationen umfassend beschreiben zu können. Daraus entstand der Standard for the Exchange of Product Model Data (STEP), der 1988 als Normenreihe 10303 veröffentlicht wurde. Getrieben von der kontinuierlichen Evolution des Computer Aided Designs wurden Mechanismen gesucht, die das dauerhafte Speichern von Konstruktions- und Produktdaten ermöglicht (Vajna et al., 2018). Eastman und Augenbroe (1998) propagierten dabei insbesondere die Möglichkeit zum Informationsaustausch innerhalb einer Disziplin, aber für den Informationsaustausch über Unternehmens- und Disziplingrenzen hinaus.

Die Werke in ISO 10303 umfassen Definitionen für die computerinterpretierbare Repräsentation von Produktdaten und deren Austausch. Hierfür wird ein Rahmenwerk beschrieben, das Produktdaten adäquat erfasst und deren systemunabhängigen Informationsaustausch ermöglichen kann (ISO, 1994a). Wenngleich die Norm über 200 einzelne Teile hat, sind für den weiteren Verlauf insbesondere die folgenden Teile relevant:

- Teil 11 spezifiziert die EXPRESS Modellierungssprache.

- Teil 21 legt dar, wie konkrete Instanzen des MOF-M0 Levels als STEP-P21 Datei kodiert werden.
- Teil 44 erläutert die XML-basierte Speicherung von Produktinformationen.

Die Modellierungssprache EXPRESS stellt Definitionen zur widerspruchsfreien Beschreibung von Informationen und Bedingungen bereit. Dazu zählen Datentypen sowie Zwangsbedingungen, die für ausgewählte Datentypen gelten (ISO, 1994b). EXPRESS ist damit zur MOF-M1 Schicht zu zählen.

EXPRESS verwendet den Begriff der *Entität* (oder im Englischen *Entity*), um Informationen von ähnlicher Objekte als Klasse zu beschreiben. Die konkrete Befüllung wird als *Instanz einer Entität* (oder *Entity Instance*) bezeichnet. Eine Entität setzt sich dabei aus einer Sammlung von Attributen zusammen, die sich über ihren Namen und einen für den erwarteten Wert passenden Datentyp definieren. Als einfache Datentypen stehen *NUMBER*, *REAL*, *INTEGER*, *STRING*, *BOOLEAN*, *LOGICAL* und *BINARY* zur Verfügung. Darüber hinaus beschreibt Teil 11 der Normenreihe diverse Aggregationsmöglichkeiten, zu denen die Datentypen *ARRAY*, *LIST*, *BAG* sowie *SET* zählen. Auf Grundlage der vorangegangen genannten Basisdatentypen können zusätzlich eigene Datentypen für die spezifische Domäne definiert werden, in der das Produktdatenmodell agiert. Neben den objektorientierten Aspekten kann EXPRESS auch bedingt prozedurale Elemente abbilden. Zu diesen gehören einfache Funktionen, Schleifen und bedingte Anweisungen. Diese Konzepte sind aber im Vergleich zu vollwertigen objektorientierten Programmiersprachen deutlich beschränkt (Weise, 2006, S. 12).

Algorithmus 2.2 beinhaltet ein Beispiel eines einfachen Datenmodells, welches in der EXPRESS Sprache notiert ist.

Algorithmus 2.2: Beispiel eines in EXPRESS formulierten Datenmodells

```
SCHEMA PointLine

ENTITY Punkt
    XWert  : REAL;
    YWert  : REAL;
END_ENTITY

ENTITY Strecke
    Start : Punkt;
    Ende  : Punkt;
END_ENTITY

END_SCHEMA
```

Neben den umfangreichen Definitionen von Datentypen und Aggregationen können Beziehungen zwischen Entitäten abgebildet werden. Als Besonderheit - und damit Konzept, das in XML und JavaScript Object Notation (JSON) nicht verfügbar ist - sei darüber hinaus das Prinzip der inversen Attribute genannt. Diese explizite Benennung von ”Rückwärts-Referenzen” ermöglicht die zusätzliche Bereitstellung einer Relationsbezeichung, die

invers zur eigentlichen Beziehung fungiert. Verfügbar ist außerdem das Prinzip der Spezialisierung beziehungsweise Vererbung zwischen Entitäten, bei der die Attribute des Supertyps ebenfalls im Subtyp vorhanden sind. Die EXPRESS-Modellierungssprache verfügt damit - wie auch XSD - über die wesentlichen beschreibenden Aspekte der objektorientierten Modellierung.

Die Beschreibung konkreter Instanzen, die den Definitionen eines in EXPRESS formulierten Datenmodells folgen, ist in Teil 21 und Teil 44 geregelt (ISO, 2016, 2019a). Die gebräuchlichste Form ist derzeit das in Teil 21 erläuterte STEP Clear Text Encoding, welches auch als STEP Physical File (SPF) bekannt ist.

Algorithmus 2.3 zeigt Instanzen im Clear-Text-Encoding nach ISO 10303-21, die das in Algorithmus 2.2 gegebene Datenmodell als Grundlage heranzieht und in Abb. 2.5 beschriebene Situation erfasst:

Algorithmus 2.3: Beispiel eines STEP-P21 basierten Informationsaustauschs

```
ISO-10303-21;

HEADER;
FILE_DESCRIPTION(('Linienbeispiel'), '2;1');
FILE_NAME('PointLine.spf','2023-05-24T08:34:45',('Sebastian Esser'),
          ('TUM CMS'),'','Autorensoftware','Adresse');
FILE_SCHEMA(('PointLine'));
ENDSEC;

DATA;
#1 = PUNKT(2.0, 1.0)
#2 = PUNKT(5.0, 2.0)
#3 = STRECKE(#1, #2)
#4 = STRECKE(#2, #5)
#5 = PUNKT(6.0, 5.0)
ENDSEC;

END-ISO-10303-21;
```

Entgegen der hierarchischen Dokumentstruktur von XML-Dokumenten werden in einer SPF Datei die einzelnen Instanzen sequenziell Zeile für Zeile innerhalb der *DATA*-Sektion aufgelistet. Jede Zeile wird dabei mit einer numerischen Entitätennummer eingeleitet. Somit können einzelne Instanzen auf andere verweisen, indem sie an der passenden Stelle als Attributwert die Entitätennummer des adressierten Objekts vorhalten.

Um auf Instanzdaten mithilfe von Quellcode zuzugreifen, die in Form STEP Physical Files vorliegen, gibt zwei wesentliche Ansätze, die als *Early Binding* und *Late Binding* bekannt sind (Amann et al., 2021; Nour & Beucke, 2008). Bei Verwendung des Early Binding Ansatzes werden zu Beginn passende Klassen und Datentypen in der gewünschten Programmiersprache erzeugt, die den Definitionen des Datenmodells in der EXPRESS-Sprache entsprechen. Die Bindung in dem EXPRESS-Schema definierten Entitäten an eine Zielsprache erfolgt zur Übersetzungszeit (Compile-Zeit). Zusätzlich müssen passende

Funktionen implementiert werden, die die in Clear-Text-Encoding gegebenen Instanzen passend in die definierten Klassen übersetzen können und auch die Serialisierung von Instanzen in die P21-Darstellung ermöglichen. Sofern sich das Datenmodell in seiner Definition verändert, so muss auch der Mapping-Prozess in die Zielprogrammiersprache wiederholt werden. Dennoch lässt sich als essentieller Vorteil des Ansatzes festhalten, dass mögliche Fehler bereits zum Zeitpunkt des Kompilierens erkannt werden. Im Gegensatz dazu erfordert die Verwendung eines Late Binding Ansatzes kein gesamtheitliches Übersetzen aller Entitäten des Datenmodells in Klassen der gewünschten Zielsprache. Stattdessen wird das Standard Data Access Interface (SDAI) verwendet, das in seiner abstrakten Form in ISO 10303-22 beschrieben ist. Diese Schnittstelle stellt Methoden zum dynamischen Lesen und Schreiben von Instanzen in eine P21-Datei bereit. Mögliche Verletzungen der im Datenmodell verankerten Vorgaben zur Nutzung von Attributtypen oder Beziehungen zwischen Instanzen können demnach erst zur Ausführungszeit festgestellt werden, da an zahlreichen Stellen der Schnittstelle mit einfachen String-Literalen gearbeitet wird. Late Binding Implementierungen sind hingegen einfacher zu handhaben, sofern sich das Datenmodell häufiger verändert. In diesem Fall muss im Gegensatz zu Early Binding Ansätzen kein sprachenspezifischer Quellcode für die Reflektion der in einer SPF-Datei enthaltenen Informationen angepasst werden.

JavaScript Object Notation (JSON)

Nach der umfangreichen Verwendung von XML entwickelte sich in den frühen 2000er Jahren mit der JSON ein weiterer Standard, der textbasierten Informationsaustausch ermöglicht. Bei JSON handelt es sich um eine Notation, die für einen möglichst einfachen und flexiblen Datenaustausch gedacht ist. Die Ursprünge liegen dabei in der Programmiersprache *ECMAScript*, der Syntax an sich ist unabhängig und wird in einer Vielzahl von Programmiersprachen unterstützt (ECMA International, 2017). Der anzuwendende Syntax ist im ECMA Standard 404 dokumentiert und umfasst lediglich Anforderungen zur formalen Struktur. Eine Spezifikation der einzelnen semantischen Objekte (wie es mit XSD möglich ist) erfolgt hingegen nicht und muss in jedem Informationsaustausch individuell zwischen den Beteiligten vereinbart werden. Mit *JSON Schema* gibt es aber Bestrebungen, die Form solcher Dateien weitergehend zu beschreiben und deren Konsistenz zu erhöhen [2].

JSON verwendet vier einfache Datentypen *string*, *numbers*, *booleans* und *null* sowie zwei Strukturen (*object* und *array*) (Crockford, 2006). Die Typen *string* und *numbers* werden in den Standards weitergehend spezifiziert und können verschiedene Formatierungen annehmen. Ein *object* stellt eine unsortierte Menge von Name-Wert-Paaren dar, wobei die Namen auf einer Hierarchieebene eindeutig sein müssen. Der Wert kann entweder einem der genannten Datentypen entsprechen oder eine der beiden Strukturen abbilden, womit eine beliebige Verkettung von Objekten möglich ist. Die Menge eines Objekts wird durch geschweifte Klammern abgeschlossen. Mehrere Instanzen innerhalb eines Arrays werden durch ein einfaches Komma getrennt, die Abgrenzung mehrerer Name-Wert-Paare

[2] https://json-schema.org/(Letzter Zugriff: 13.01.2024)

innerhalb eines Objekts werden durch Semikolons abgegrenzt. Ein Doppelpunkt realisiert die Zuweisung zwischen Name und Wert.

Gemäß Erläuterungen im ECMA Standard und RFC 4627 wird die UTF-8 Kodierung als Standard zur Kodierung erwartet. Es sind aber auch die 16 und 32 bit Varianten möglich. Eine Spezifizierung erfolgt über die ersten zwei Bits der Textdatei (Crockford, 2006).

Algorithmus 2.4: Einfaches JSON Beispiel zur Beschreibung der in Abb. 2.5 illustrierten Domäne

```
{
  "Linienbeispiel": [
    {
      "Strecke": {
        "Start": {
          "XWert": 2.0,
          "YWert": 1.0
        },
        "Ende": {
          "XWert": 2.0,
          "YWert": 1.0
        }
      }
    },
    {
      "Strecke": {
        "Start": {
          "XWert": 5.0,
          "YWert": 2.0
        },
        "Ende": {
          "XWert": 6.0,
          "YWert": 5.0
        }
      }
    }
  ]
}
```

Ontologien

Neben den beschriebenen Strategien zur Beschreibung von Objektmodellen haben parallel umfangreiche Enwicklungen von computerlesbaren Ontologien stattgefunden. Allgemein wird der Ontologie-Begriff als Sammlung von Vokabular verstanden, das zur Beschreibung einer Domäne eingesetzt werden kann (Busse et al., 2014; Uschold & Gruninger, 1996). Das zentrale Ziel ist - wie auch bei den zuvor vorgestellten Auszeichnungssprachen - eine computerlesbare Darstellung von Information und deren Wiederverwendbarkeit an anderen Stellen. Dabei spielt die Vernetzung über das Internet eine entscheidende Rolle (Berners-Lee et al., 2001), wodurch umfangreiche Schlussfolgerungen in den verknüpften

Datensätzen möglich werden. Besonders relevant ist heute die Web Ontology Language (OWL), auf deren Charakteristika später in Abschnitt 3.2.1 detailliert eingegangen wird.

2.3.5 Produktdatenmodelle zur Beschreibung gebauter Umwelt

Wie bereits in den Ausführungen zum Begriff des Modells in Abschnitt 2.3.1 dargelegt, kann (und muss) ein Datenmodell immer dem Zweck einer bestimmten Abstraktion genügen beziehungsweise einen gezielten Nutzen verfolgen. Dieses Phänomen trifft auch auf Fragestellungen zu, die die Informationserfassung über bauliche Anlagen betreffen. Diese Art von Modellbildung wird auch als *Produktdatenmodell* bezeichnet und hat zum Ziel, einen Anwendungsbereich möglichst vollständig und standardisiert zu beschreiben (M. Fischer & Froese, 1992).

Turk beschreibt in seinen Veröffentlichungen, dass die Modellbildung immer subjektiver Wahrnehmung unterliegt (Turk, 1999, 2001). Für die qualitative Einordnung eines Datenmodells führen Hartmann et al. (2017) drei Kategorien aus dem Bereich des allgemeinen Informationsmanagements und der Ontologie-Entwicklung ein, die den Zweck von Informationsmodellen beschreiben können. Sie beurteilen Datenmodelle nach der die Fähigkeit, (i) einen Sachverhalt semantisch adäquat darzustellen, (ii) der konzeptionellen Vollständigkeit der Darstellung sowie (iii) der Komplexität, die mit der Implementierung des Datenmodells in ein Software-Systemen einhergeht. Darüber hinaus kommen Hartmann et al. (2017) zu dem Schluss, dass Verbesserungen in einem der drei Qualitätskriterien meist zu Lasten der anderen beiden Kriterien führen. Die Indikatoren verhalten sich daher nach ihren Ausführungen konträr zu einander. Hartmann et al. (2017) folgern daraus, dass es kein Datenmodell für die Beschreibung baulicher Anlagen geben kann, welches für alle Interessensgebiete und Anwendungsszenarien gleichermaßen geeignet ist.

Für die Beschreibung von Produkten, Objekten und Vorgängen im Kontext des Bauwesens existieren daher folgerichtig heute zahlreiche Applikationen, die BIM-Modelle erstellen und verarbeiten können und deren Informationsaustausch mittels geeigneter Austauschstandards unterstützen. Für die verschiedenen Fachdisziplinen haben sich im Laufe der Zeit teils hochspezialisierte Anwendungen entwickelt, die die Erstellung notwendiger Fachplanungen bestmöglich unterstützen. Solche Programme verwenden in der Regel proprietäre, interne Datenstrukturen, die nur in begrenztem Umfang für Endnutzer zugänglich sind (zumeist nur durch die Verwendung von API-Schnittstellen). Man spricht in diesem Zuge auch von *Closed-BIM* (Borrmann et al., 2021b). Erschwerend kommt hinzu, dass die programmspezifischen Repräsentationen der erstellten BIM-Modelle zumeist nicht oder nur zwischen Softwareprodukten desselben Herstellers kompatibel austauschbar sind. Um dennoch Planungsinformation modellbasiert zwischen Softwareprodukten unterschiedlicher Hersteller auszutauschen, wurden verschiedene Datenmodelle entwickelt, die dem *Open-BIM* Gedanken des interdisziplinären, modellbasierten Informationsaustausch verfolgen. Orthogonal zum Aspekt des Informationsaustauschs über verschiedene Fachapplikationen hinweg wird die Nutzung über verschiedene Aufgaben und Phasen hinweg betrachtet. So bezeichnet *Little-BIM* die Erhebung und Verwendung von Information

in einem BIM-Modell, die lediglich innerhalb eines Anwendungsszenarios weiterverwendet werden sollen. Im Gegensatz dazu versteht man und *Big-BIM* die Nutzung erhobener Informationen über verschiedene Phasen, Anwendungsfälle und Disziplinen hinweg.

Die für das Bauwesen etablierten Datenmodelle bedienen sich den zuvor eingeführten allgemeinen Auszeichnungssprachen wie XML und XSD oder der Kombination aus EXPRESS und STEP-P21 nach ISO 10303 und sind teilweise auch als Ontologie-Repräsentation verfügbar. Sie verfügen über umfangreiche Objekt- und Attributdefinitionen für Objekte der gebauten Umwelt und werden quell-offen sowie frei zugänglich dokumentiert. Die wichtigsten Datenmodelle, die insbesondere für die Modellierung von Gebäuden, Infrastrukturanlagen und ganze Städte eingesetzt werden können, werden im Folgenden kurz erläutert.

Industry Foundation Classes

Die Industry Foundation Classes wurden von der international agierenden Organisation BuildingSMART International (bSI) geschaffen und stellen ein weltweit akzeptiertes Datenmodell zum Austausch von BIM-Modellen in herstellerneutraler Weise dar. Das Datenmodell liegt seit 2005 als ISO-Standard vor. Dieser wurde vom Europäischen Komitee CEN/TC 442 als Europäische Norm übernommen und hat darüber hinaus auch den Rang einer deutschen Norm erlangt. Das Datenmodell bietet umfangreiche Möglichkeiten zur geometrischen und semantischen Modellierung von Bauwerken und verfügt zudem über ein umfangreiches Portfolio an Beziehungsklassen zur Beschreibung von Abhängigkeiten zwischen Objekten. Zudem bestehen Möglichkeiten, eigene benutzerdefinierte Attributsätze an Bauteile anzuhängen, womit ein BIM-Modell bei Bedarf mit nicht-standardisierten Eigenschaften angereichert werden kann. Hervorzuheben sind darüber hinaus die umfangreichen Möglichkeiten der geometrischen Modellierung, die von einfachen Oberflächenbeschreibungen bis hin zu komplexen prozeduralen Darstellungen reichen.

Das Datenmodell kann als Grundlage für verschiedenste Simulationen und Analysen im Kontext des Building Information Modelings herangezogen werden und ist nach wie vor Gegenstand der Forschung. Braun (2020) nutzte beispielsweise Industry Foundation Classes (IFC)-basierte BIM-Modelle für die Baufortschrittsüberwachung auf Baustellen. Preidel (2020) demonstrierte, wie IFC-Daten für Belange der automatisierten Prüfung von BIM-Modellen hinsichtlich geltender Normen und Richtlinien verwendet werden können. Auch Forth et al. (2023) verwenden das IFC-Modell zur Beschreibung von BIM-Modellen und werten die enthaltenen Informationen hinsichtlich Nachhaltigkeitsaspekten in frühen Entwurfsphasen aus. Amann (2018) diskutierte zudem die Erweiterung des IFC-Datenmodells um weitere funktionale Auszeichnungen und schlägt die Implementierung prozeduraler Strukturen vor, um den Implementierungsaufwand sowie die konsistente Interpretation der verarbeitenden Systeme zu verbessern. In Slepicka et al. (2022) sind Ansätze dokumentiert, inwiefern die Strukturen des IFC-Datenmodells auch für Prozesse der additiven

Fertigung geeignet sind und welche Erweiterungen notwendig wären, um Fabrikationsinformationen konsistent zu beschreiben.

Bis zur Version 4 fokussierte sich der IFC-Standard vor allem auf Aspekte von Hochbauvorhaben. Da mit den Stufenplänen zur Einführung von BIM insbesondere öffentliche und halb-öffentliche Auftraggeber zu herstellerneutralen Ausschreibungen und Vergabeverfahren verpflichtet sind, wird seit 2011 intensiv an den Erweiterungen des Datenmodells für Anwendungsfälle der Infrastrukturplanung gearbeitet. Sowohl auf nationaler als auch internationaler Ebene existierten hierzu verschiedene Teilprojekte, die sich unter anderem mit der Erweiterung des Datenmodells aber auch mit der passenden Abbildung länderspezifischer Aspekte befassten (siehe hierzu auch König et al., 2017). Im internationalen Kontext sind dabei insbesondere die Erweiterungsprojekte IFC-Bridge, IFC-Road, IFC-Rail und IFC-PortsAndWaterways zu nennen, deren gemeinsame Grundlagen im Projekt „IFC Overall Architecture“ definiert wurden (Jaud et al., 2020). Nach den Kleinversionen IFC4x1 (erste Unterstützung von Trassierungsinformationen) (Amann et al., 2014), IFC4x2 (Erweiterung um brückenspezifische Komponenten) (Borrmann et al., 2019) und mehreren Iterationen der IFC4x3 Version wurde Mitte 2021 die Version IFC4x3_RC4 als *Candidate Standard* auf internationaler Ebene verabschiedet und dem Prozess zur Aktualisierung des zugehörigen ISO-Standards zugeführt. Die sich derzeit in Entwicklung befindliche Erweiterung des Datenmodells zu Aspekten des Grund- und Tunnelbaus wird voraussichtlich als Version IFC4x4 bezeichnet (Borrmann et al., 2022).

Das IFC-Datenmodell wurde bisher vor allem als EXPRESS Repräsentation (ISO, 1994b) veröffentlicht. Die Beschreibung von MOF-M0 Instanzdaten erfolgt zumeist als STEP-P21 Datei, die den Konventionen von ISO (2016) folgt. In den letzten Jahren zeichnet sich nunmehr eine zunehmende Öffnung in Richtung der W3C Standards ab, um Kompatibilität und Akzeptanz in anderen Branchen zu erhöhen (van Berlo et al., 2021).

GML, CityGML und LandXML

Als Datenmodell für die Abbildung großskaliger Stadtmodelle konnte sich in den letzten Jahren zunehmend der CityGML-Standard etablieren, der Definitionen und Objektbeschreibungen urbaner Strukturen für unterschiedlichste Anwendungen und Zwecke ermöglicht (Ebertshäuser et al., 2021). Die Einsatzbereiche des Datenmodells umfassen Visualisierungen großskaliger Strukturen und semantischer Untersuchungen bis hin zu komplexen numerischen Simulationen im Bereich der Verkehrs- und Überflutungsuntersuchungen. Methoden und Datenmodelle für diese Zwecke werden auch unter dem Begriff Geographisches Informationssystem (GIS) zusammengefasst.

CityGML baut auf die in ISO 19136 (ISO, 2020) beschriebene herstellerneutrale Auszeichnungssprache der Geographic Markup Language auf (nicht zu verwechseln mit Generalized Markup Language (GML)), die als Basissprache wiederum auf XSD zur Beschreibung des Datenmodells und XML zur Serialisierung der Instanzen zurückgreift. Die Geographic Markup Language stellt Begriffe zur Modellierung geografischer Strukturen und deren gegenseitigen Wechselwirkungen bereit. Diese können in sogenannten

Anwendungsschemata weitergehend für spezifische Anwendungsfälle spezifiziert werden. Bekannte Vertreter sind hierbei das eben benannte CityGML, aber auch die Normbasierte Austauschschnittstelle (NAS) und Keyhole Markup Language (KML). Alle Datenmodelle sind als freie GIS-Standards weit verbreitet und werden unter dem Dach des Open Geospatial Consortium (OGC) weiter ausgebaut (Kutzner et al., 2020).

Neben CityGML wurden insbesondere für die Beschreibung von Straßeninfrastruktur und den zugehörigen Trassierungs- und Ausstattungselementen umfangreiche Datenmodelle entwickelt, deren historische Entwicklung durch Rebolj et al. (2008) zusammengefasst wurden. Daraus entwickelte sich der XML-basierte LandXML-Standard, der mit Version 1.1 und 1.2 insbesondere ab 2010 weite Verbreitung in verschiedenen Softwareprodukten fand. Amann (2018) berichtet allerdings, dass die Version 2.0 aufgrund fehlender Organisation und Unstimmigkeiten innerhalb der Definitionen nur noch spärlichen Einsatz fand. Das Open Geospatial Consortium hat sich diesen Problemen mittlerweile angenommen und LandXML unter dem Namen LandInfra bzw. InfraGML [3] als offiziellen OGC-Standard weiterentwickelt.

RailML und PlanPro

Das PlanPro-Datenmodell soll Planern die durchgehende, elektronische Datenhaltung bei der gesamten Planung von Leit- und Sicherungstechnik (LST) ermöglichen, die im Laufe der Planung eines Elektronischen Stellwerks entstehen. Konkret bedeutet dies, dass bereits erarbeitete Daten weitergegeben und eigene Ergebnisse in bestehenden Projekten ergänzt werden können. Langfristiges Ziel ist die Konzeption einer Datenstruktur, in die alle Informationen über bestehende sowie neu geplante und ausgeführte LST-Komponenten gespeichert werden (Buder, 2017). Neben der konsistenten Datenhaltung in der Planungsphase ist es das erklärte Ziel, die erzeugten Daten in den Betrieb elektronischer Stellwerke zuzuführen und damit einen durchgehend digitalen Prozess von der Planung bis zum Betrieb und der Steuerung von schienengebundenem Verkehr zur Verfügung zu stellen.

PlanPro ist ein herstellerunabhängiges Datenformat und wird seit 2008 erarbeitet. In diesem Datenmodell werden zahlreiche Informationen über Komponenten erfasst, die für die Organisation des Zugverkehrs notwendig sind und die Grundlage für moderne elektronische Stellwerke bilden. Das Ziel ist die konsistente Datenweitergabe im Planungsprozess bis hin zur Anwendung der Daten im Stellwerk.

Neben PlanPro mit starkem Fokus auf die Belange der Leit- und Sicherungstechnik in Deutschland, existiert mit dem RailML Standard im internationalen Kontext ein weiteres Datenmodell, welches spezifische Belange des schienengebundenen Verkehres beschreibt. Im Vordergrund stehen dabei die Teilbereiche *Interlocking* zur Beschreibung von Signaltechnik, *Infrastructure* für Belange des Schienennetzes, *Rolling stock* zur Repräsentation verschiedener Fahrzeuge und *Timetable* zur Modellierung von Zeit- und Fahrplänen. Ergänzt wird das Datenmodell zudem durch den *Common* Bereich, welches Abhängigkeiten

[3] https://www.ogc.org/standard/infragml/ (letzter Zugriff: 11.01.2024)

zwischen den einzelnen Teilbereichen des Datenmodells definiert (Nash et al., 2010). Die topologische Struktur des Streckennetzes wird zudem durch das RailTopoModel beschrieben (Wunsch & Jaekel, 2017).

In beiden Fällen werden Instanzdaten mit XML beschrieben, die Datenmodelle sind als Sammlungen von XSD-Dokumenten veröffentlicht.

2.3.6 Zusammenfassung Produktdatenmodelle im Bauwesen

Wie dargestellt existieren umfassende Definitionen von Produktdatenmodellen, die verschiedene Szenarien und fachliche Informationen einzelner Disziplinen adäquat beschreiben können. Diese bedienen sich etablierter Auszeichnungssprachen, um Informationen über Objekte, ihre Zustände sowie zugehöriger Beziehungen formalisiert und unabhängig von den eingesetzten Planungswerkzeugen zu beschreiben. Die Diversität an verschiedenen Fachsichten, die innerhalb der einzelnen Datenmodelle berücksichtigt wurden, lässt erwarten, dass in Zukunft weitere Applikationen Funktionen zur Erstellung, Analyse und Integration hochstrukturierter Modellinformationen bereitstellen werden. Zugehörige Ansätze wurden beispielsweise von Breunig et al. (2017) und Beck (2023) für die Kopplung von BIM- und GIS-Datenmodellen dargelegt. Gleichzeitig erwächst aus technischer Sicht mit der Vielzahl an verschiedenen Auszeichnungssprachen und Abstraktionsannahmen der Bedarf nach Strategien, die umfängliches Informations- und Versionsmanagement über verschiedene Datenmodelle hinweg ermöglichen. Abb. 2.6 fasst schematisch die Vielfalt zusammen, die im Folgenden für Fragen der modellbasierten Zusammenarbeit und des interdisziplinären Datenaustauschs zu berücksichtigen sind.

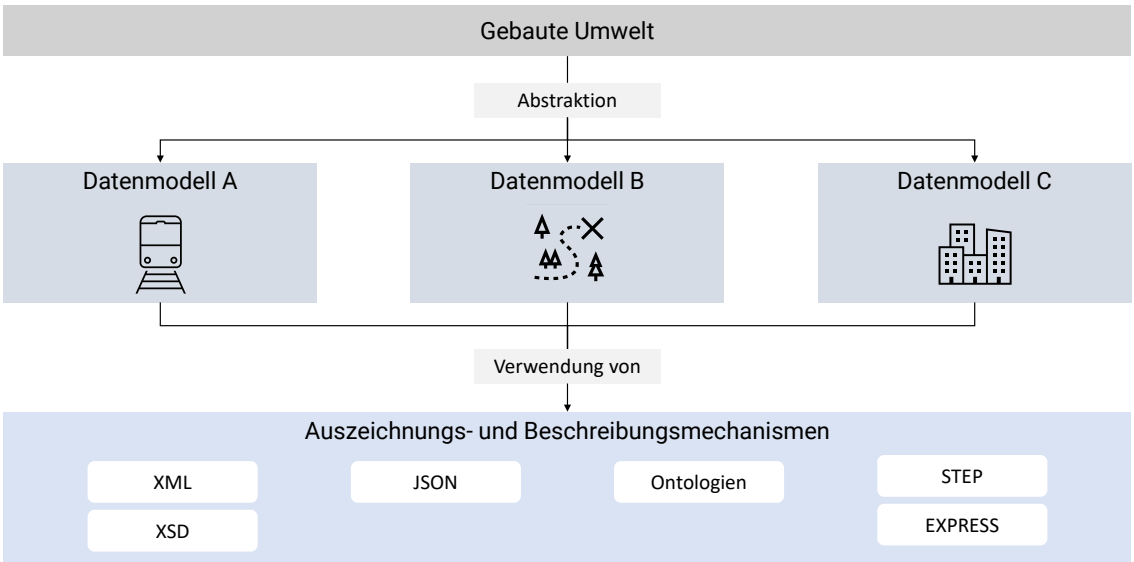


Abbildung 2.6: Zusammenhang zwischen der gebauten Umwelt, deren Abstraktion in verschiedenen Datenmodellen und verfügbarer Serialisierungstechnologien

2.4 Methoden der modellbasierten Kollaboration

Nach Borrmann et al. (2021b) und Eastman et al. (2011) sind durch den Austausch und die Nutzung von Modellen Effizienzsteigerungen für alle an Planungs- und Ausführungsprozessen beteiligten Akteuren zu erwarten. Obwohl jeder Beteiligte individuelle Ziele und Fachsichten verfolgt, die mithilfe eines BIM-Modells beschrieben werden sollen, so hat sich dennoch gezeigt, dass die gesamtheitliche Betrachtung und Analyse mehrerer Modelle innerhalb eines koordinierten Gesamtmodells wichtige Informationen über die Gesamtplanung und möglichen Konflikten liefern kann. Es wird dabei in Kauf genommen, dass Zwischenstände der einzelnen Modelle möglicherweise inkonsistent oder sogar konträr zu einander sein können. Solche Probleme können schlussendlich nur durch die Anwendung von durchgängigem Datenaustausch aufgedeckt und anschließend iterativ gelöst werden, um zu definierten Zeitpunkten eine im Gesamten widerspruchsfreie Planung zu erzielen.

Im Folgenden werden nun wesentliche Grundlagen der modellbasierten Zusammenarbeit in Entwurf, Planung, Ausführung und Betrieb im Bauwesen vorgestellt und anschließend bestehende Limitationen heutiger Systeme hervorgehoben.

2.4.1 Etablierter Stand der modellbasierten Zusammenarbeit im Bauwesen

Wesentliche Aspekte zu dem Austausch von Informationsressourcen im Bauwesen sind unter anderem von Schapke et al. (2021) zusammengefasst worden. Analogien und Herausforderungen zur durchgängigen Nutzung von Produktdaten im Bauwesen wurden von Borrmann et al. (2009) erläutert. Grundsätzlich bieten sich für die modellbasierte Zusammenarbeit im Bauwesen in verteilten, asynchronen Situationen zwei wesentliche Ansätze an, die auf den in Eastman (1999) erläuterten Szenarien aufbauen.

Gemeinsame Zusammenarbeit in einem zentral gespeicherten Modell
Dieser Ansatz geht von einem zentral gespeicherten Gesamtmodell aus, welches von den verschiedenen am Projekt beteiligten Parteien gemeinsam genutzt wird. Sämtliche Planungen und Änderungen werden direkt in diesem Modell umgesetzt, sodass eine sofortige Synchronisierung mit allen verbundenen Nutzern stattfinden kann. Um ambivalente Änderungen durch mehrere Nutzer zu verhindern, werden in der Regel *Locking*-Mechanismen eingesetzt, die bestimmte Teile des Modells vor Änderungen durch andere schützt. Ein wesentlicher Vorteil des Ansatzes ist die direkte Verfügbarkeit aller Änderungen für alle Projektbeteiligten.

Lose Zusammenarbeit mit Teil- und Fachmodellen und deren Koordination in Gesamtmodellen zu spezifischen Zeitpunkten
Im Kontrast zu zentral gespeicherten Modellen erarbeiten in diesem Ansatz alle Domänen eigenständige *Teil-* und *Disziplinmodelle* in Softwareprodukten ihrer Wahl und stellen diese anschließend zu koordinativen Zwecken zur Verfügung. Zu vorab definierten Zeitpunkten werden diese Modelle dann zentral verwaltet, sodass ein koordiniertes Gesamtmodell entsteht. Das Gesamtmodell dient wiederum als Grundlage für interdisziplinäre Über-

prüfungen wie Kollisionschecks oder Visualisierungen. Als zentralen Vorteil erhält diese Form der Kooperation die klare Zuordnung der Verantwortungsbereiche. Zudem können Zuständigkeiten und Autorenrechte bei Änderungen eindeutig identifiziert werden.

Nicht zuletzt aus der Erkenntnis, dass eine allgemeine, umfassende Definition aller Bedarfe der an einem Projekt beteiligten Gewerke in einem gemeinsamen Modell a-priori schwer bis unmöglich erreichbar ist (Willenbacher, 2002), hat sich in den letzten Jahren das föderale Arbeiten mit Disziplinmodellen durchgesetzt. Frühe Überlegungen hierzu sind unter anderem durch Kiviniemi et al. (2005) dokumentiert worden. Für die Modellkoordination in einem verteilten System kommen im BIM Reifegrad 2 sogenannte gemeinsame Datenumgebungen (engl. Common Data Environment (CDE)) zum Einsatz. Diese werden im ISO-Standard 19650 wie folgt definiert:

> *„source of information for any given project, used to collect, manage and disseminate all relevant approved project documents for multi-disciplinary teams in a managed process“* CEN, 2018

Eine CDE-Plattform ist als gemeinsamer digitaler Projektraum bekannt, der klare Statusdefinitionen und Statusübergänge für ausgetauschte Datensätze bietet. Zusätzlich können verschiedene Freigabe- und Genehmigungsprozesse eingesetzt werden, die an den Schnittstellen verschiedener Statuszustände eingesetzt werden. Außerdem verfügen diese Plattformen über unterschiedliche Zugriffsbereiche für die verschiedenen Projektbeteiligten (Preidel et al., 2021). Damit unterstützen diese die von Schapke et al., 2021 beschriebenen kollaborativen Ansätze mit passender Technologie, um den Datenaustausch zwischen verschiedenen Projektbeteiligten umzusetzen.

Gegenüber Dokumentmanagement-Systemen und einfachen Netzwerkspeichern grenzen sich die CDE-Plattformen insbesondere durch die auf das Bauwesen zugeschnittenen Funktionalitäten ab, die unter anderem in Deutsches Institut für Normung e. V (2019a, 2019b) weitergehend spezifiziert sind und beispielsweise Anforderungen zur Nutzerverwaltung, Zu den Kernfunktionen moderner CDE-Plattformen gehören integrierte Viewer-Funktionalitäten sowie die Möglichkeit, verschiedene Informationsquellen integriert anzeigen zu können (beispielsweise durch Überlagerung von Plänen und Modell-Schnitten). Zudem verfügen zahlreiche moderne Plattformen über Mechanismen für eine modellzentrierte Kommunikation durch das BIM Collaboration Format (BCF). Damit können Mängel und Hinweise direkt einzelnen Bauteilen eines Modells zugeordnet werden und herstellerneutral ausgetauscht werden.

Jede an einem Projekt beteiligte Disziplin arbeitet dabei mit für sie passenden Applikationen und stellt für andere Planer relevante Informationen in Form von BIM-Modellen (im Sinne der MOF-M0 Ebene) bereit. Das von einer Disziplin erstellte Modell enthält fachliche Informationen ebendieser Disziplin und wird konsequenterweise als Fach- oder Disziplinmodell bezeichnet (Schapke et al., 2021). Fachmodelle werden zu vorab definierten Zeitpunkten durch die Modellautoren auf eine CDE-Plattform geladen. Modelle bilden zusammen mit begleitenden Dokumenten wie Detailzeichnungen, Bauteillisten oder sonstiger begleitender Dokumentationen einen Informationscontainer, der anschließend

als kleinste monolithische Einheit auf der Projektplattform verarbeitet wird. Nach den Definitionen in ISO 19650-1 wird allgemein angenommen, dass innerhalb eines Informationscontainers mehrere Dateien übermittelt werden. Als Dokument wird demnach auch ein einzelnes BIM-Modell angesehen, es können aber auch andere digital repräsentierte Informationsarten wie Zeitpläne, Kostenübersichten oder Mängeldokumentationen übermittelt werden.

Für die kollaborative Verwendung von Informationscontainern definieren ISO 19650 (CEN, 2018) und PAS 1192 (British Standards Institution, 2013) mehrere Zustände, die unter anderem in Preidel et al. (2021) ausführlich erläutert werden:

- **in Bearbeitung/work in progress**: Enthaltene Informationen sind noch nicht finalisiert und nicht geprüft.
- **geteilt/shared**: Die Informationen sind finalisiert, wurden aber noch nicht auf Kompatibilität und Integrität zu anderen geteilten und veröffentlichten Informationen geprüft.
- **veröffentlicht/published**: Die Informationen im Informationscontainer sind überprüft und sind belastbar.
- **archiviert/archived**: Der Informationscontainer wurde in das Archiv verschoben (zumeist aufgrund einer neueren Version, die die bisherigen Daten ersetzt).

Auf die technische Umsetzung dieser Merkmale wird in ISO 19650 nur lose eingegangen. In der Regel erfolgt die Implementierung aber heutzutage auf Webservern, die je nach Hersteller über Webbrowser oder Betriebssystem-Integrationen erreicht werden können. Nour (2009) beschrieb früh eine Client-Server-Architektur, bei der die verschiedenen Disziplinen in einem Projekt als Clients fungieren. Dabei kann innerhalb jedes Clients der Austausch disziplin-interner Dokumente und Modelle stattfinden. Sobald ein teilbarer Zustand erreicht ist, erfolgt der Upload der Fachinformationen auf den zentralen Server. Optional verfügt jeder Client über eine als *Workbench* bezeichnete Umgebung, die Methoden zur passenden Aufbereitung von IFC-Daten in ein relationales Datenbanksystem bereitstellt. Dies vereinfacht den Informationsimport in nachgelagerte Systeme. Je nach technischen Schnittstellen sieht Nour aber auch eine direkte Kommunikation mit dem zentralen Modellserver vor. Abb. 2.7 zeigt die von Nour vorgeschlagene Systemarchitektur.

2.4.2 Limitationen bestehender modellbasierter Kollaborationssysteme

Verschiedene Veröffentlichungen der letzten Jahre haben die praktische Anwendung von CDE-Plattformen kritisch beleuchtet. Diese zeigen diverse Unzulänglichkeiten auf, die in der derzeitigen Umsetzung kollaborativer Prinzipien nach ISO 19650 zu finden sind. So berichtet Jaskula et al. (2023) von technischen Limitationen in verschiedenen CDE-Systemen, die BIM Level 2 Prinzipien implementieren. Awwad et al. (2022) haben die Anwendung der BIM-Stufen 1 und 2 in kleinen und mittelständischen Unternehmen

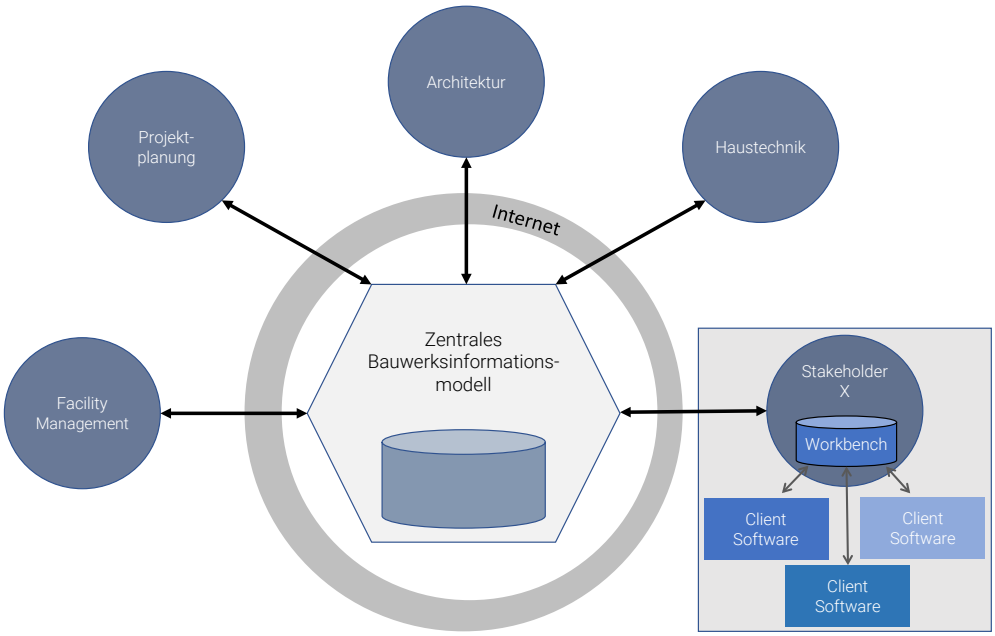


Abbildung 2.7: Kollaborationsarchitektur nach Nour (2009)

(KMU) im Vereinigten Königreich im Rahmen einer umfangreichen Literaturrecherche und Experteninterviews analysiert. Sie kamen zu dem Schluss, dass das wahre Potenzial von BIM-bezogenen Techniken von KMUs im britischen Baugewerbe noch nicht vollständig ausgeschöpft wurde. Attrill und Mickovski (2020) sowie Alankarage et al. (2022) berichten ebenfalls über verschiedene Bedenken und Mängel, die aus Sicht von Endanwendern die Einführung von BIM Level 3 Konzepten behindern und nach wie vor die umfängliche Anwendung von BIM Level 2 Prinzipien zu Problemen in der Praxis führt. Ihre Ergebnisse basieren auf Interviews und zeigen auf, dass Konzepte, die gemeinhin als BIM Level 2 verstanden werden, immer noch nicht in allen Teilen der AEC-Branche vollständig übernommen wurden. Insbesondere wird festgestellt, dass an BIM-Prozessen beteiligte Personen nach wie vor nicht über ausreichende Qualifikationen verfügen, um BIM-Prinzipien in vollem Umfang auszuführen. Darüber hinaus betonen sie das Fehlen einer konsistenten Kontrolle zwischen Modellen und 2D-Zeichnungen, die aus diesen abgeleitet wurden. Wenn Änderungen an einem Modell vorgenommen werden, werden die Pläne oft nicht entsprechend aktualisiert. Ähnliche Beobachtungen wurden auch von Kurul et al. (2016) berichtet. Gemäß ihrer Literaturrecherche und Fallstudien wurde die Einführung der modellbasierten Zusammenarbeit auf dem britischen Markt nicht etabliert, obwohl öffentliche Verwaltungen und große Auftragnehmer ihre Anforderungen erweitert haben.

Neben den Herausforderungen, die sich durch das Erzeugen und das Verarbeiten ausgetauschter Modelle ergeben, sind zusätzlich Fragestellungen zu berücksichtigen, die sich aus dem Wunsch nach Informationsintegration aus verschiedenen Informationsquellen ergeben. Bucher und Hall (2020) haben hierfür eine Terminologie vorgeschlagen, um verschiedene Dimensionen der Interoperabilität im Kontext von CDE-Systemen zu unterscheiden. Gemäß ihren Ausführungen können aktuelle CDE-Systeme auf einer eindimensionalen Interoperabilitätsebene gesehen werden, da die Systeme den Datenaustausch

mittels BIM-Modellen ermöglichen, aber in der Regel begrenzte Möglichkeiten haben, zusätzliche Informationen damit zu verknüpfen. Die zweidimensionale Interoperabilität zeichnet sich durch die Einführung von Austauschschnittstellen zwischen CDE-Systemen aus, während die dreidimensionale Interoperabilität diese Idee durch die Verbindung von betriebswirtschaftlichen Plattformen mit verschiedenen CDE-Systemen erweitert.

Die Studien verdeutlichen, dass nach Auffassung der Autorinnen und Autoren CDE-Systeme nach wie vor nicht in voller Funktionalität in Projekten eingesetzt werden, da die individuellen Anforderungen einzelner Beteiligter, die angebotenen Funktionalitäten in den jeweiligen Produkten oder technische Limitationen hemmend auf die durchgängige Nutzung solcher Plattformen wirken. Insbesondere wird hervorgehoben, dass CDE-Plattformen häufig nach wie vor als einfache Netzwerkspeicher angesehen werden und passende Arbeitsabläufe nicht implementiert werden. Zudem müssen Integritätsprüfungen veränderter Daten häufig nach wie vor manuell vorgenommen werden, da die lose Kopplung der einzelnen Modelle im Koordinationsmodell keine direkte Aussage über mögliche Auswirkungen oder konkurriert veränderter Information in verschiedenen Fachmodellen zulässt.

Aus den aufgeführten Arbeiten lassen sich die folgenden Nachteile und Forderungen nach Verbesserungen der derzeitigen Situation zusammenfassen:

Fehlende Kopplung von mehreren Versionen eines Modells

Die Zusammenhänge zwischen zwei Modellen und deren möglicher Versionen werden innerhalb einer Projektplattform derzeit nicht explizit genug abgebildet. Ausgewählte Produkte bieten zwar Visualisierungsmöglichkeiten zum Ein- und Ausblenden spezifischer Modellversionen an, allerdings kann darüber keine computerlesbare Information ermittelt werden, die eine weitere Automatisierung ermöglichen würde. Eingrenzend wird angenommen, dass die Interaktion mit geometrisch-semantischen Modellen weiter an Bedeutung gewinnen wird. Die Kopplung zwischen abgeleiteten Plänen und zugehörigen Modellen wird hingegen nicht im Detail untersucht. Hierzu sei der Leser beispielsweise auf die Arbeit von Weidinger (2022) verwiesen.

Fehlende Kopplung zwischen Modellen mit verschiedenen Disziplin-Sichten

Wie in Abschnitt 2.3.5 dargestellt gibt es zahlreiche Produktdatenmodelle, die verschiedene Sichten auf reale Objekte anbieten. Derzeit werden im Zuge eines Datenaustauschprozesses mögliche Zusammenhänge zwischen diesen Modellen nicht weiter analysiert. Daraus wird gefolgert, dass es neben einer Verfolgung der Änderungen zwischen zwei Modellen auch einer Analyse zwischen Modellen verschiedener Disziplinen bedarf, um mögliche implizite Auswirkungen direkt in den Versionskontrollprozessen abbilden zu können.

Aus diesen Erkenntnissen lässt sich ableiten, dass insbesondere die Interpretation eines Informationscontainers als monolithische Datei einen großen Teil des Potentials kollaborativer Ansätze ungenutzt lässt. Bisher muss jede neue Version alle vorgesehenen Überprüfungen erneut durchlaufen, auch wenn in der Regel nur Teilbereiche eines Dokuments oder eines Modells bearbeitet werden.

2.5 Versionskontrollsysteme

Versionskontrollsysteme ermöglichen die konsistente Verwaltung von sich über einen Zeitraum verändernden Dateien. Zugleich verfügen sie über Funktionalität, um zu jedem Zeitpunkt zu einer spezifischen Version zurückzuspringen (Chacon, 2009). Solche Systeme werden insbesondere im Bereich der Softwareentwicklung umfangreich eingesetzt. Das Hauptziel eines Versionskontrollsystems besteht dabei vor allem in der Bereitstellung einer gemeinsam nutzbaren Plattform zum Austausch von Quellcode sowie zugehöriger Protokollierung der getätigten Entwicklungsprozesses und zugehöriger Entscheidungen. Sie stellen daher eine potenzielle Technologie zur Lösung der zuvor beschriebenen Unzulänglichkeiten der bisherigen Praxis im Bauwesen dar.

2.5.1 Historische Entwicklung

Frühe Ansätze der Versionsverwaltung gehen auf Rochkind (1975) zurück, der in den 1970er Jahren zur Verwaltung von Quellcode veröffentlichte. Dabei beschreibt Rochkind die Herausforderung, dass Software stetigen Änderungen unterworfen ist und somit stets neue Versionen entstehen, die sich anschließend auf verschiedenen Speichermedien befinden können. Als Lösungsvorschlag schlägt er den Einsatz von diskreten *Deltas* vor, die die Änderung zwischen zwei Zuständen eines Moduls bzw. des Quellcodes im Allgemeinen erfassen. Mehrere Entwicklungsschritte führen konsequenterweise zur Formulierung mehrerer Deltas, die wiederum eine Entwicklungskette bilden. Durch Interpretation dieser Kette kann das *Source Code Control System (SCCS)* jeden in der Entwicklungskette erfassten Zustand reproduzieren. Da die Versionskontrolle lediglich lokal erfolgte, sind diese Systeme auch als *Local Version Control Systems* oder dem Akronym *Revision Control System* bekannt. Die Arbeiten von Tichy (1984) leisteten einen maßgeblichen Beitrag zur kontinuierlichen Weiterentwicklung und fanden später Einzug in das GNU Betriebssystem. Primär wurde das *Revision Control System* für die Versionskontrolle von Textdateien ausgelegt, später aber für verschiedene Dateiarten wie Skizzen, Bildern und Dokumenten eingesetzt. Die Beschreibung der Versionsinkremente erfolgte dabei zeilenweise. Hierbei wurde dokumentiert, welche Zeilen gelöscht, geändert oder hinzugefügt wurden und welchen Inhalt sie haben. Sollten lediglich Teile einer Zeile modifiziert worden sein, wurde ebenfalls die gesamte Zeile ausgetauscht. Für die Ermittlung der Änderungen referenzierte Tichy den *diff*-Algorithmus von Hunt und Mcilroy (1976). Ein detailliertere Einführung in *diff*-Algorithmen für Textdateien gibt Abschnitt 2.5.2.

Mit der Einführung von dieser frühen Versionskontrollsysteme war zwar eine Versionierung auf lokaler Ebene möglich, aber noch kein kollaborativer Informationsaustausch über mehrere Arbeitsplätze hinweg erschaffen. Mit dem Wachstum von Softwareprojekten und der Notwendigkeit der Zusammenarbeit entstanden schließlich zentralisierte Systeme wie CVS (Concurrent Versions System) und später Subversion (SVN). Die Kernidee hinter diesen Ansätzen besteht in der Vorhaltung des Repositories auf einem zentralen Server. Einzelne Clients konnten Dateien *auschecken*, bearbeiten und anschließend wieder auf

dem zentralen Server *einchecken*. Damit wurden erste Implementierungen angewandt, die eine prägnante Administration über Zugriffsrechte und Änderungsmanagement ermöglichte. Kritisch zu bewerten war jedoch die Konzentration des gesamten Informationsflusses auf einen zentralen Server. Tritt am Server ein Problem auf, so können auch die Nutzer keine Daten mehr aus- oder einchecken. Wie so oft motivierte dieser Nachteil eine Weiterentwicklung der Systeme hin zu *distributed version control systems*. Das bekannteste und heute weit verbreitete System ist Git, welches 2005 durch Linus Torwalds entwickelt wurde. Die zentrale Errungenschaft liegt an der Übertragung der gesamten Repositories auf jeden Client. Damit bilden die Clients indirekt auch gleich eine Art Sicherung der serverseitigen Inhalte ab, da jeder Clone-Prozess tatsächlich ein vollwertiger Backup-Schritt des aktuellen Serverzustands ist. Zudem bedeutet das Vorhalten einer vollwertigen Kopie, dass Nutzer auch offline uneingeschränkt an dem Projekt arbeiten können.

Abb. 2.8 veranschaulicht das allgemeine Vorgehen in Git in einem lokalen Kontext. Git überwacht Dateien, die in einem definierten Arbeitsverzeichnis abgelegt werden. Um den Zustand einer Datei in einem Versionsschritt zu sichern, wird die Datei zu erst in die *Staging area* hinzugefügt. Dies geschieht durch den *add*-Befehl. Alle Dateien in der *Staging Area* können anschließend mit dem Befehl *commit* im Versionsschritt gesichert werden. Neben der Verwendung als Funktionsbezeichnung wird der Begriff auch für die in einem Versionsschritt gespeicherten Informationen genutzt.

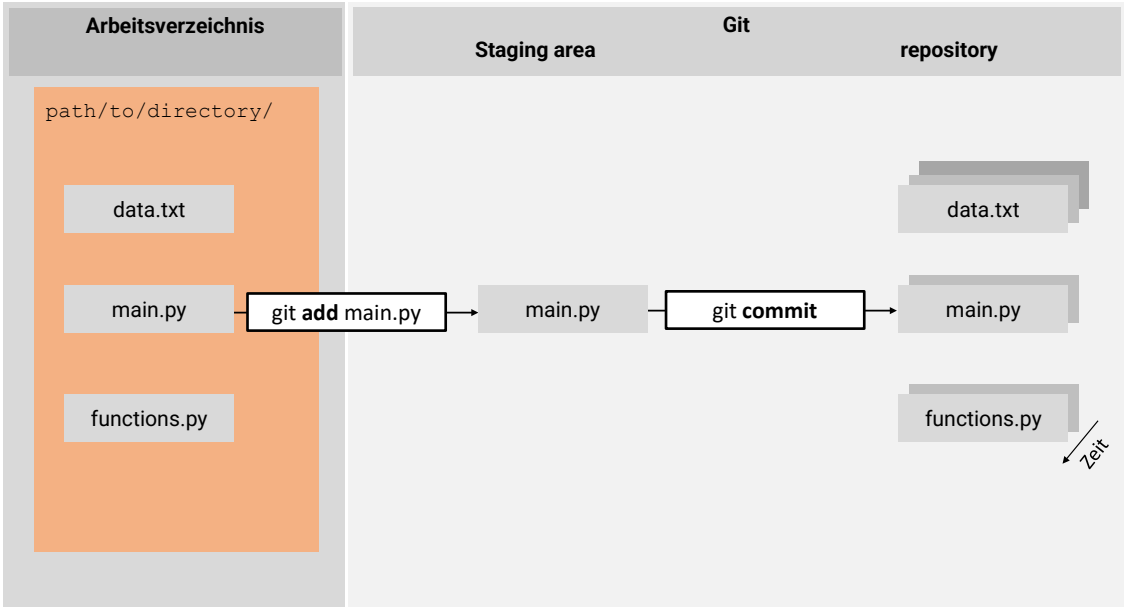


Abbildung 2.8: Git Workflow (angelehnt an Blischak et al. (2016))

Im Gegensatz zu seinen Vorgängern speichert Git nicht nur einzelne Inkremente zwischen zwei Versionsschritten, sondern in jedem Versionsschritt einen Snapshot des Projekts, was das Rückverfolgen und Verzweigen deutlich erleichtert (Chacon, 2009, S. 12). Bei einem Snapshot werden alle Dateien innerhalb des Repositories berücksichtigt. Sofern an einer Datei nichts verändert wurde, wird lediglich eine Referenz zur vorherigen (identischen) Datei gespeichert. Im Falle einer Änderung erfasst Git den neuen Zustand der Datei, sodass zukünftige Dateien wieder darauf referenzieren können. Zudem sind große

Sprünge und Vergleiche zwischen einzelnen Versionen deutlich performanter realisierbarer, da beispielsweise eine vor kurzem veränderte Datei sehr zügig mit einem beliebigen Zustand in der Vergangenheit verglichen werden kann. Anstatt eine lange Änderungshistorie zu rekonstruieren, kann Git die Verweise der entsprechenden Datei rückverfolgen und schließlich eine Diff-Operation auf den gewünschten Versionszuständen ausführen. Jeder Versionsschritt wird mit einer Hashsumme versehen, die aus dem Inhalt des Snapshots errechnet wird. Somit kann jeder Versionsschritt eindeutig durch die Hashsumme adressiert werden.

Im weiteren Verlauf werden wesentliche Prinzipien zum *Diff* und *Patch* von unstrukturierten und ausgezeichneten Textdateien dargestellt.

2.5.2 Diff und Patch für unstrukturierte Textsequenzen

Unter dem Begriff `diff` wird die Ermittlung der maximalen übereinstimmenden Submenge zwischen zwei Sequenzen von Textzeichen verstanden (Myers, 1986). Aus technischer Sicht wird eine möglichst minimale Prozedur bestehend aus Entfernen-, Modifkations- und Einfügen-Operationen gesucht, die eine Ausgangssequenz in einen neuen Zustand transformiert.

Zur Beschreibung der Änderungen nutzte Tichy (1984) dabei die Operatoren *Löschen (d)*, *Modifizieren (c)* und *Anhängen (a)*. Die Mengensymbole $<$ und $>$ werden hierzu ergänzend verwendet, um den zu löschenden und einzufügenden Inhalt in Textform darzustellen. Der Abgleich zweier Textmengen erfolgt im Allgemeinen zeilenweise, da nach den Ausführungen von Tichy (1984) eine Sequenz von Änderungsoperatoren auf Basis einzelner Textzeichen deutlich länger in der Ausführung benötigen würde und nicht signifikant kürzer wäre.

Als Beispiel zur Beschreibung der Änderungen werden vereinfacht die beiden folgenden Zeichensequenzen aus der Veröffentlichung von Hunt und Mcilroy (1976) betrachtet, die jeweils eine Textdatei darstellen. Aus Gründen der Übersichtlichkeit werden die Zeilen in Algorithmus 2.5 und Algorithmus 2.6 horizontal illustriert.

Algorithmus 2.5: Textsequenz 1	Algorithmus 2.6: Textsequenz 2
A B C D E F G	W A B X Y Z E

Eine mögliche Sequenz an Änderungsoperationen kann sich für die gegebenen Sequenzen wie folgt darstellen:

1. Hänge nach Zeile 0 an: *W*
2. Modifiziere Zeilen 3-4 (bisher: *C D*) in die neuen Zeilen 3-5 (neu: *X Y Z*)
3. Lösche Zeilen 6-7 (bisher: *F G*)

Als kompakte Darstellung wurden die Änderungsoperatoren mit einem einzelnen Buchstaben abgekürzt. Daraus resultiert die in Algorithmus 2.7 gezeigte Darstellung. Die Zeilen 1-2 beschreiben das Anhängen von W nach der 0. Zeile der Ausgangssequenz. Des weiteren beinhalten die Zeilen 3-9 das Ersetzen der Sequenz $C\ D$ durch $X\ Y\ Z$ und die Zeilen 10-12 das Löschen der Zeichen $F\ G$.

Algorithmus 2.7: Sequenz der Änderungsoperationen, um die Zeichenkette $ABCDEFG$ in $WABXYZE$ umzuformen

```
1   0 a 1,1
2   > W
3   3,4 c 4,6
4   < C
5   < D
6   ---
7   > X
8   > Y
9   > Z
10  6,7 d 7
11  < F
12  < G
```

Aus mengentheoretischer Sicht versuchen *diff*-Algorithmen die längste gemeinsame Sequenz zwischen zwei zu vergleichenden Textsequenzen zu finden, um anschließend eine möglichst kurze Darstellung der notwendigen Änderungsoperatoren zu ermitteln. Hunt und Mcilroy (1976) beschreiben hierfür das als *Hunt–Szymanski Algorithmus* bekannte Vorgehen zur Suche der längsten gemeinsamen Sequenz. Ausgangspunkt sind zwei Textsequenzen A und B. Die größte Zeichenmenge, die beiden Textsequenzen gemein ist, ermittelt sich anschließend nach Gleichung (2.1).

$$P_{ij} = \begin{cases} 0\ , wenn\ i = 0\ oder\ j = 0 \\ P_{i-1,j-1} + 1\ ,\ wenn\ A_i = B_j \\ max(P_{i-1,j}, P_{i,j-1})\ , wenn\ A_i \neq B_j \end{cases} \tag{2.1}$$

Die Funktionsweise lässt sich anschaulich durch die Notation in einer tabellarischen Form wie in Abb. 2.9 darstellen. Die Sequenz A wird horizontal aufgetragen, wohingegen die in B enthaltenen Elemente vertikal notiert werden. Beispielhaft werden die beiden Textsequenzen $A = ABCABBA$ und $B = CBABAC$ betrachtet. Abb. 2.9 zeigt die resultierende Matrix für die Sequenzen gemäß Gleichung (2.1).

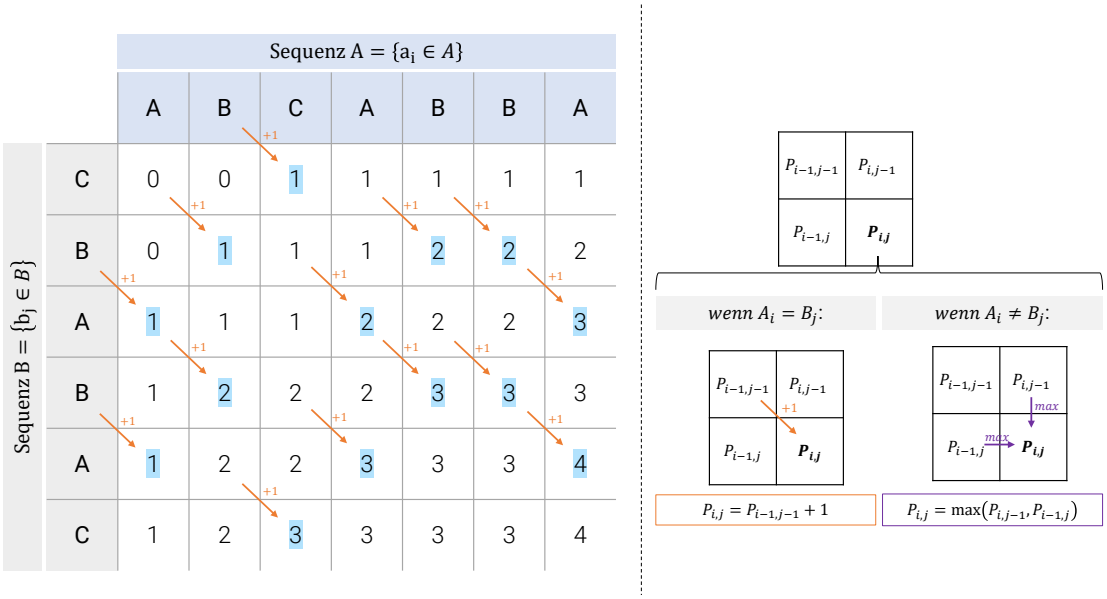


Abbildung 2.9: Hunt–Szymanski Algorithmus nach Hunt und Mcilroy (1976)

Für jeden Kandidat $P_{ij}|i = 0..m, j = 0..n$ wurde die oben genannte Berechnungsvorschrift ausgewertet. Die gesuchte längste gemeinsame Sequenz ergibt anschließend aus den grün hervorgehobenen P_{ij}. Damit werden mehrere Möglichkeiten ermittelt, woraus jenes Ergebnis mit der maximalen Anzahl an Einträgen ausgewählt wird. Im illustrierten Beispiel sind unter anderem die folgenden Sequenzen als Kandidaten erkennbar:

1. $C \rightarrow B$
2. $C \rightarrow B \rightarrow A$
3. $B \rightarrow A$
4. $B \rightarrow A \rightarrow B \rightarrow A$
5. $A \rightarrow B \rightarrow A$
6. $A \rightarrow C$
7. $C \rightarrow A \rightarrow B \rightarrow A$

Als längste gemeinsame geordnete Menge wurden demnach die Abfolgen $B\ A\ B\ A$ (Nr. 4) sowie $C\ A\ B\ A$ (Nr. 7) gefunden. Die zugehörigen Änderungsskripte weisen folgerichtig die gleiche Länge auf.

Laut Myers (1986) existieren neben den von Hunt und Mcilroy (1976) dokumentierten Ansätzen weitere Algorithmen, die eine möglichst große Menge an Zeichen ermitteln und für die folgende Formulierung eines Änderungsskripts genutzt werden können. Er selbst stellt einen verbesserten Algorithmus vor, der nach seinen Ausführungen eine deutlich bessere Laufzeitkomplexität gegenüber den Ansätzen von Hunt und Mcilroy

(1976) erzielt. Myers Algorithmus wird in der *diff*-Funktion von Unix-basierten Systemen eingesetzt. Dieser basiert auf der Konstruktion des so genannten *Edit-Graphs*, mit dem die maximal übereinstimmende Sequenz bestimmt werden soll. Er hat zum Ziel, eine möglichst effiziente Variante zur Beschreibung des Unterschieds zwischen zwei geordneten (Zeichen-)Mengen zu ermitteln. Abb. 2.10 zeigt die Konstruktion des Graphs für die Sequenzen $A = ABCABBA$ und $B = CBABAC$.

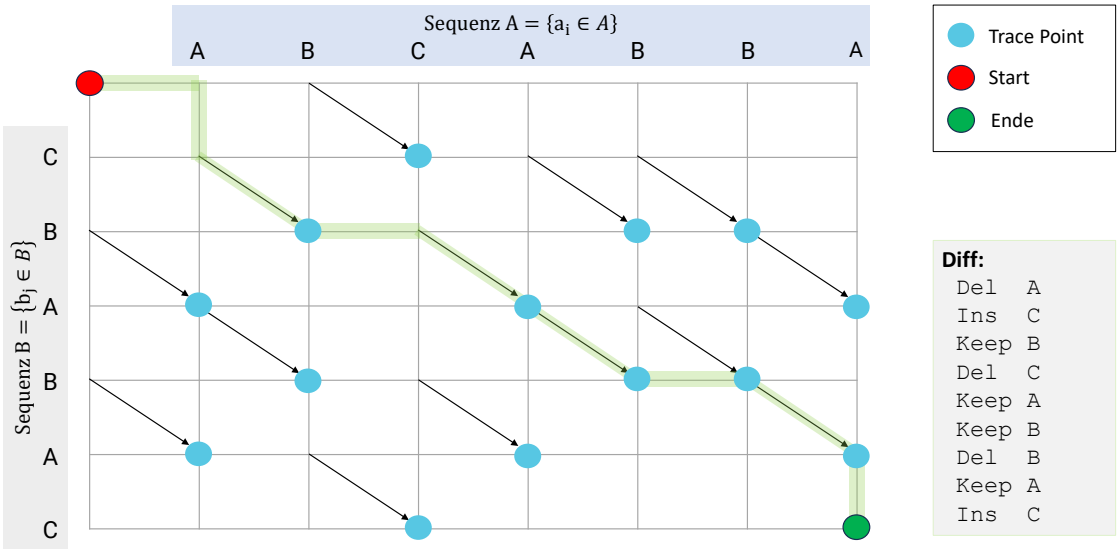


Abbildung 2.10: Edit-Graph zur Ermittlung der maximalen gemeinsamen Subsequenz zweier Zeichenfolgen (angelehnt an Myers (1986)).

Die Sequenz A wird erneut horizontal aufgetragen, wohingegen die in B enthaltenen Elemente vertikal notiert werden. Zwischen zwei aufeinander folgenden Elementen einer Sequenz wird eine gerichtete Kante eingefügt. Im nächsten Schritt werden nun die *Trace Points* markiert (in blau dargestellt). Bei diesen Knoten handelt es sich um Positionen, bei denen das Zeichen $a \in A$ mit dem Zeichen $b \in B$ übereinstimmt. Zusätzlich wird zu jedem *Trace Point* eine diagonale Kante eingeführt.

Nun kann die Suche nach möglichen Diff-Prozeduren beginnen. Es gilt dabei die Konvention, dass der konstruierte Graph nur nach rechts, nach unten oder diagonal navigierbar ist. Das Folgen einer horizontalen Kante beschreibt dabei das Löschen des Zeichens, auf das die aktuelle Kante zeigt, aus der initialen Sequenz A. Im Gegensatz dazu bedeutet das Folgen einer vertikalen Kante das Hinzufügen des Zeichens aus Sequenz B, auf das die Kante zeigt. Die Einfügeoperation ist nach der Indexposition des ausgehenden Knotens durchzuführen.

Zur Beschreibung der Änderung zwischen beiden Sequenzen existieren mindestens $m \times n$ Möglichkeiten, wobei m die Länge der Sequenz A und n die Länge der Sequenz B beschreiben. Die kombinatorische Mindestanzahl tritt ein, wenn keinerlei übereinstimmenden Elemente in beiden Mengen vorhanden sind (also keine diagonalen Kanten und keine `Match Points`). Mit jedem Match-Point erhöht sich die Anzahl der Diagonalen um 1 und

damit auch die Anzahl möglicher Pfade. Um nun die maximal übereinstimmende Teilmenge zwischen den Sequenzen A und B zu bestimmen, muss lediglich jener Pfad gewählt werden, der die maximale Anzahl an Diagonalen beinhaltet. In Abb. 2.10 ist in grün die für die gegebenen Zeichensequenzen optimale Übereinstimmung markiert. Die triviale Lösung durch vollständiges Löschen und Einfügen der jeweiligen Sequenzen ergibt sich aus der Navigation entlang der äußeren horizontalen und vertikalen Kanten.

Aus den erläuterten Algorithmen zur Bestimmung einer maximalen gemeinsamen Sequenz zweier Zeichenketten wird ersichtlich, dass es in der Regel mehrere Lösungen gibt, die die Transformation eines Ausgangs- in einen modifizierten Zustand beschreiben können. Würde man beispielsweise eine beliebige Sequenz verwenden, die man mit den Algorithmen von Hunt und Mcilroy (1976) oder Myers (1986) ermittelt hat, so könnte man in allen Fällen eine passende Folge an Änderungsprozeduren formulieren, die den Ausgangszustand des Textes in den gewünschten Zielzustand überführen würde. Selbst bei der Verweigerung, eine gemeinsame Sequenz zwischen Start- und Zielzustand zu berücksichtigen, könnten eine Abfolge von Änderungsprozeduren formuliert werden, die die Transformation passend durchführt (z.B. zuerst alle Elemente des Zielzustands einfügen und anschließend alle Elemente der Startsequenz entfernen). Dieser Fall ist in Abb. 2.10 in blau markiert. Natürlich führt die Auswahl der *maximalen* gemeinsamen Sequenz zu einer *minimalen* Änderungsprozedur, was aus Gründen der Effizienz anzustreben ist. Wenngleich die Ermittlung von Änderungsskripten für Textsequenzen noch keine direkt anwendbare Lösung für die Versionsüberwachung von BIM-Modellen bereitstellt, verdeutlichen die erläuterten Verfahren aber bereits die Herausforderung, dass in der Regel mehr als eine mögliche Lösung zur Ermittlung einer Transformationsvorschrift existiert, sofern lediglich der initiale und der gewünschte Zielzustand einer Zeichenmenge bekannt sind. Dieser Gedanke wird später für die Ermittlung von Modifikationen in BIM-Modellen nochmals aufgegriffen.

Die erläuterten Algorithmen ermöglichen die Ermittlung einer möglichst großen Teilmenge zweier gegebenen Sequenzen. Die Resultate der Verfahren können in der Folge zur Formulierung notwendiger Transformationen herangezogen werden, um die vorgenommene Änderung zu übertragen. Dieser Gesamtprozess wird auch als *diff* bezeichnet und unter anderem in Unix-basierten Betriebssystemen unter gleichem Namen als Funktion vorgehalten. Als Gegenstück gibt es auch einen *patch* Befehl. Diese Funktion erwartet als Eingabe die veraltete Textdatei sowie Transformationsvorschrift. Anschließend wendet dieser Befehl die Änderungsoperatoren auf die veraltete Sequenz an, um den gewünschten Zielzustand zu erreichen.

2.5.3 Diff und Patch für XML und JSON

Die rudimentären Funktionalitäten, die mit `diff` und `patch` für beliebige Zeichensequenzen erschaffen wurden, ebneten den Weg für komplexere Überlegungen zur Behandlung hierarchischer Daten. Chawathe et al. (1996) beschreiben einen Ansatz zur Ermittlung von *Deltas* in hierarchisch strukturierten Daten. Folglich beschreiben die Autoren einen

Algorithmus, um eine möglichst optimale Darstellung der Änderungslogik zwischen zwei hierarchisch organisierten Datenstrukturen zu finden. Sie interpretieren die Datenmenge dabei als hierarchische Struktur, die mengentheoretisch einen sogenannten *Baum* darstellt. Ein Baum setzt sich aus einer Sammlung von Knoten zusammen. *Oberknoten* bezeichnet dabei einen Knoten, der anderen Knoten übergeordnet ist. Im Umkehrschluss beschreibt der Begriff *Unterknoten* Elemente, die sich hierarchisch unterhalb eines Knotens befinden. Genauere Informationen zu den mathematischen Konzepten und den zugrundeliegenden mengentheoretischen Grundlagen werden in Kapitel 3 ausführlich erläutert.

Als Basisoperationen sehen Chawathe et al. (1996) folgende vier Operatoren vor, um die Änderungen zur Umformung eines Initialzustandes in den gewünschten Zielzustand für Baumstrukturen zu erreichen:

- **Hinzufügen eines Unterknotens**: Diese Operation wird als $INS((x, l, v), y, k)$ notiert, wobei (x, l, v) den neu einzufügenden Knoten x mit Label l und Wert v beschreibt, y den neuen Oberknoten und k die Position des neuen Kindes unter dem Oberknoten beschreibt.
- **Entfernen eines Blattknotens**: Mit der Operation $DEL(x)$ wird das Löschen des Knotens x beschrieben. Nur Knoten, von denen keine weiteren Unterknoten abhängig sind, können gelöscht werden. Sofern ganze Teil-Bäume entfernt werden sollen, so müssen entweder zuerst alle Unterknoten entfernt oder der Teilbaum an einen anderen Knoten verschoben werden.
- **Modifizieren eines Knotens**: Die Operation $UPD(x, val)$ beschreibt die Änderung eines Wertes v am Knoten x. Diese Operation verändert den Baum nur semantisch, nicht aber in seiner Topologie.
- **Verschieben eines Unterbaums**: Notiert als $MOV(x, y, k)$ beschreibt diese Operation das Verschieben des Unterbaums ab Knoten x an einen anderen Knoten y. Knoten x wird dort als k-ter Unterknoten eingehängt.

Aufgrund der hierarchischen Abhängigkeiten ist es relevant, dass die Operationen in der festgelegten Reihenfolge ausgeführt werden. Gleichzeitig soll eine möglichst *günstige* Sequenz von Modifikationen gefunden werden. Chawathe et al. (1996) definieren hierfür Kostenfunktionen und definieren den Aufwand für das Löschen (Delete), Einfügen (Insert) und Verschieben (Move) mit den Kostenwerten $c_D(x) = c_I(x) = c_M(x) = 1$. Weiter erläutern sie den möglichen Aufwand, der zur Identifikation und dem Anwenden von Attributänderungen notwendig sind. In der Regel treten die oben benannten Operationen in Kombination auf. Daher argumentieren sie im weiteren, dass es bei umfangreicheren semantischen und topologischen Änderungen sinnvoll sein kann, Knoten nicht erst zu verschieben und dann zu modifizieren, sondern im Falle großer semantischer Änderungen Knoten lieber mit einer Kombination aus Löschen und Einfügen in der Sequenz der notwendigen Änderungsoperationen zu erfassen.

Um die minimale Änderungsvorschrift bestehend aus den oben genannten Basisoperatoren beschreiben zu können, muss analog zum Vorgehen bei unstrukturierten Textsequenzen zuerst die maximale Übereinstimmung der beiden untersuchten Versionen gefunden werden, um anschließend die notwendigen Transformationen in Form eines *edit scripts* ermitteln zu können. Dies wird mit einer Breitensuche auf den beiden Baumstrukturen realisiert. Im wesentlichen werden bei jedem Rekursionsschritt der Tiefensuche die direkten Unterknoten sowie deren Position verglichen. Da per Definition die Entfernen-Operation lediglich Knoten an den Baumenden löschen kann, müssen diese Knoten nach der Tiefensuche weitergehend prozessiert werden. Zudem werden in der Veröffentlichung weitere Ausführungen zur Optimierung des beschriebenen Verfahrens skizziert, die an dieser Stelle nicht weiter erläutert werden.

Die grundlegenden Operationen, die von Chawathe et al. (1996) beschrieben wurden, haben auch die Definitionen verschiedener Konzepte beeinflusst, die durch die Internet Engineering Task Force zu inkrementellen Verfahren für Auszeichnungssprachen veröffentlicht werden.

In dem durch die IETF diskutierte RFC 6902 wird ein Verfahren beschrieben, das partielles Aktualisieren von JSON-Daten ermöglicht (Bryan & Nottingham, 2013). In einem *Patch* werden alle Operationen erfasst, die für die Aktualisierung eines bestehenden JSON Objekts notwendig sind. Zur Beschreibung der einzelnen Modifikationen werden in RFC 6902 die folgenden Operationen definiert:

- *remove*: Löscht die Struktur unterhalb des Elements.
  ```
  { "op": "remove", "path": "/biscuits"}
  ```
- *insert*: Fügt eine gegebene Struktur unterhalb des Elements ein.
  ```
  { "op": "add", "path": "/biscuits/1", "value": { "name": "Ginger Nut" } }
  ```
- *replace*: Ersetzt eine bestehende Struktur.
  ```
  { "op": "replace", "path": "/biscuits/0/name", "value": "Chocolate Digestive" }
  ```
- *copy*: Kopiert ein Objekt von einem gegebenen Element *from* in den neuen, in *path* spezifizierten Ast.
  ```
  { "op": "copy", "from": "/biscuits/0", "path": "/best_biscuit" }
  ```
- *move*: Verschiebt ein Objekt von einem gegebenen Element *from* in den neuen, in *path* spezifizierten Ast. Dieser Operand führt zum gleichen Ergebnis wie die Kombination aus *remove* und *add*.
  ```
  { "op": "move", "from": "/biscuits/0", "path": "/best_biscuit" }
  ```

Neben diesen Operatoren gibt es noch einen weiteren Operator, der für die Überprüfung von Pfaden genutzt wird. Er überprüft, ob ein bestimmter Wert an einer durch einen Pfad beschriebenen Stelle vorhanden ist. Für alle Methoden werden die für die Operation relevanten Elemente durch eindeutige Zeiger (engl. *Pointer*) adressiert. Diese sind in RFC 6901 ausführlich definiert worden (Bryan et al., 2013).

Beispielhaft sei nun eine Modifikation des in Algorithmus 2.4 gezeigten JSON Objekts illustriert, die den in Abb. 2.5 gezeigten Sachverhalt zur Beschreibung zweier Strecken in Auszügen beschreiben.

Algorithmus 2.8: Initiales JSON

```
{
  "Linienbeispiel": [
    {
      "Strecke": {

        "Start": {
          "XWert": 2.0,
          "YWert": 1.0
        },
        "Ende": {
          "XWert": 2.0,
          "YWert": 1.0
        }
      }
    },
    {
      "Strecke": {
        "Start": {
          "XWert": 5.0,
          "YWert": 2.0
        },
        "Ende": {
          "XWert": 6.0,
          "YWert": 5.0
        }
      }
    }
  ]
}
```

Algorithmus 2.9: Modiziertes JSON

```
{
  "Linienbeispiel": [
    {
      "Strecke": {
        "Name": "Strecke 1",
        "Start": [
          2.0,
          1.0
        ],
        "Ende": [
          2.0,
          1.0
        ]
      }
    }
  ]
}
```

Visuell wird bereits bei der Gegenüberstellung beider Objekte ersichtlich, dass ein neues Attribut *Name* unter $Strecke$ eingefügt wurde. Zudem wurde das zweite Listenelement von $Linienbeispiel$ gänzlich entfernt. Schließlich wurden auch noch die Beschreibungen des Start- und Endpunkts der ersten Linie modifiziert. Das zugehörige Patch-Objekt ist in Algorithmus 2.10 dargestellt.

Algorithmus 2.10: Beschreibung eines JSON-Patches zur Transformation des in Alg. 2.9 gegebenen Objekts in die in Alg. 2.10 gegebene Form

```
[
  {
    "op": "add",
    "path": "/Linienbeispiel/0/Strecke/Name",
    "value": "Strecke 1"
  },
  {
    "op": "replace",
    "path": "/Linienbeispiel/0/Strecke/Start",
    "value": [
      2.0,
      1.0
    ]
  },
  {
    "op": "replace",
    "path": "/Linienbeispiel/0/Strecke/Ende",
    "value": [
      2.0,
      1.0
    ]
  },
  {
    "op": "remove",
    "path": "/Linienbeispiel/1"
  }
]
```

Die für JSON erläuterten Prinzipien für die Formulierung von inkrementellen Patches wurden einige Jahre zuvor bereits für XML-Dokumente in RFC 5261 beschrieben (Urpalainen, 2008). Die Grundprinzipien zum Modifizieren der baumartigen Strukturen weisen umfangreiche Ähnlichkeiten auf. Da XML allerdings über vielfältigere Beschreibungsmechanismen verfügt, gibt es für das Patching von XML-Dokumenten eine größere Zahl an Operatoren. So existieren beispielsweise verschiedene Operatoren, die das Einfügen von Attributen, das Einfügen eines gesamten Elements behandeln und das Hinzufügen eines neuen Namespaces ermöglichen. Details sind dem zugehörigen RFC-Dokument zu entnehmen und werden an dieser Stelle nicht weiter behandelt.

2.5.4 Verschmelzen divergierender Versionen

Die vorangegangenen Abschnitte fokussierten sich auf Methoden zur Abstraktion von Änderungen zwischen zwei Textsequenzen und deren spätere Anwendung auf eine veraltete Version. Diese Verfahren nehmen implizit an, dass die Abfolge von Versionsschritten chronologisch ist. Insbesondere in Systemen, die allerdings ein entkoppeltes Arbeiten erlauben, kann es je nach Nutzerverhalten zu Versionen kommen, die von verschiedenen Nutzern auf konkurrierende Weise bearbeitet wurden. Daher bedarf es nun weiterer Un-

tersuchung zu Prinzipien der Versionsverschmelzung (engl. *merging*). Allgemein befasst sich der Begriff des *Mergings* mit der Rekombination divergierender Repräsentationen, die zu einem früheren Zeitpunkt einen gemeinsamen Ursprung hatten (Mens, 2002). Änderungen an einer Datei, die sich gegenseitig nicht beeinflussen, werden dabei vom Versionskontrollsystem meist automatisch übernommen. Ein aktiver Eingriff des Nutzers ist nur dann erforderlich, wenn es konkurrierende Änderungen gibt und diese einen Konflikt verursachen.

Die zugrundeliegenden Herausforderungen wurden insbesondere bei der kollaborativen Bearbeitung formatierter Textdokumente festgestellt. Lindholm (2004) beschreiben beispielsweise einen Ansatz, um mehrere unabhängig von einander modifizierte XML-Dokumente wieder in eine gemeinsame Repräsentation zu kombinieren. Auch der Google-Entwickler Fraser (2009) beschreibt in seiner Veröffentlichung das Framework *Differential Synchronization* einen kompakten Synchronisierungsmechanismus, welcher die Unterschiede zwischen zwei Dokumentzuständen als Diff abstrahiert und diese anschließend in einem verteilten System mit allen teilnehmenden Clients synchronisiert. Als relevanten Anwendungsfall erläutert er das automatische Speichern eines Dokuments auf einem zentralen Speicherort, um verschiedene auf die Information zugreifende Geräte nahezu in Echtzeit synchron zu halten.

Die zuvor vorgestellten Initiativen verdeutlichen, wie vielfältig die Methoden zur inkrementellen Synchronisation vorgenommener Änderungen sind. Besonders durch die stetig zunehmende Nutzung verschiedener Endgeräte und verbesserter Zugriffsmöglichkeiten auf zentral gespeicherte Daten erhöht sich zunehmend der Bedarf, notwendigen Datentransfer möglichst effizient zu gestalten. Im Folgenden werden daher allgemeine Konzepte zu verteilten Systemen eingeführt sowie deren Anwendbarkeit für Repräsentationen erläutert, die im Bauwesen Anwendung finden.

2.6 Grundprinzipien von verteilten Systeme und deren Anwendung im Bauwesen

Die zuvor beschriebenen Methoden zur Ermittlung passender Änderungsoperationen zur Umformung von (strukturierten) Textsequenzen von einem Ausgangs- in einen Zielzustand wurden zu Beginn vorrangig zur Versionierung auf einem einzelnen Endgerät angewandt. Besonderes Potenzial ist aber insbesondere dann gegeben, wenn sie im Zusammenhang mit verteilten Systemen eingesetzt werden. Überlegungen zu computergestützten Kooperationssystemen finden sich in der Literatur seit den späten 1980er Jahren (Grudin, 1994).

In den folgenden Abschnitt werden zuerst technologische Grundlagen zu verteilten Systemen behandelt und anschließend deren Anwendung in Versionskontrollsystemen für Textdateien erläutert. Anschließend werden diese Prinzipien auf die Herausforderungen im Kontext von Produktdatenmodellen übertragen und Limitationen aufgezeigt, die bei dem Einsatz von textbasierten Systemen auftreten.

Die konzeptionelle Grundlage für die Überlegungen zu verteilten Systemen bilden *Transaktionen*. Transaktionen werden weitreichend implementiert, um ein System in einem konsistenten Zustand zu erhalten, und werden daher oft mit dem Begriff der *Nebenläufigkeitskontrolle* in Verbindung gebracht. In Datenbanksystemen beschreibt die Transaktion die Umformung eines Systems von seinem Ausgangszustand in einen neuen konsistenten Zustand (Haerder & Reuter, 1983). Hierfür definieren die folgenden vier ACID-Merkmale Aspekte, die eine erfolgreiche Transaktion auszeichnen:

- Die **Atomarität** (engl. *Atomicity*) beschreibt, dass alle Operationen, die zusammen eine Transaktion ergeben, ausgeführt werden müssen oder ein *Rollback* zum vorherigen Zustand erfolgt
- Das **Konsistenz-Merkmal** (engl. *Consistency*) besagt, dass jede Transaktion die Datenbank in einen neuen validen Zustand der Datenbank überführt.
- Die **Isolation** besagt, dass während der Ausführung einer Transaktion keine anderen Operationen durchgeführt werden können, die nicht zur Transaktion gehören. Somit wird sichergestellt, dass ein *Rollback* möglich ist, sofern während der Ausführung unerwartete Fehler auftreten.
- Die **Dauerhaftigkeit** (engl. *Durability*) besagt, dass Daten nach erfolgreicher Ausführung einer Transaktion garantiert und dauerhaft gespeichert sind. Dazu gehören auch etwaige Abstürze, Fehler oder sonstige Ereignisse, die während der Transaktion unerwartet auf das Speichersystem einwirken.

Obwohl diese Prinzipien weit verbreitet in vielen Computersystemen verwendet werden, existieren darüber hinaus verschiedene Anwendungsszenarien für computergestützte Systeme, die weder eine unmittelbare Konsistenz erfordern noch zu einem streng pessimistischen Nebenläufigkeitskontrollmodell passen. Um die Anforderungen an unmittelbare Konsistenz zu lockern, haben Fox et al. (1997) die BASE-Prinzipien als optimistischeres Nebenläufigkeitsparadigma eingeführt:

- **Grundlegende Verfügbarkeit** (engl: *Basic Availablity*): Die gespeicherten Daten sind während Transaktionen nicht blockiert. Daher sind die Daten die meiste Zeit verfügbar.
- **Weicher Zustand** (engl. *Soft State*): Gespeicherte Operationen müssen während der Schreiboperation nicht konsistent sein und müssen nicht direkt an alle vorhandenen Repliken des Datensatzes verbreitet werden.
- **Beliebige Konsistenz** (engl. *Eventual Consistency*): Die Konsistenz zwischen Repliken wird schließlich zu einem späteren Zeitpunkt erreicht.

Man spricht bei der Verwendung der BASE-Prinzipien auch von einer optimistischen Nebenläufigkeitskontrolle. Zugleich werden die BASE-Prinzipien auch häufig mit dem Begriff

langer Transaktionen in Verbindung gebracht. Der Ansatz nimmt billigend in Kauf, dass einzelne Bestandteile des Systems möglicherweise für einen Zeitraum nicht synchronisiert sind und Änderungen zu unspezifischen Zeitpunkten herstellen. Eilebrecht beschreibt damit einhergehende Risiken wie folgt:

> *„[Nebenläufigkeit] ist ein bisschen wie mit Zeitreisen: Solange Sie nichts anfassen und mit niemandem reden, kann eigentlich nichts schief gehen. Andernfalls können interessante Dinge passieren. “* Eilebrecht und Starke, 2019

Darüber hinaus erörtern Dayal et al. (1990) Herausforderungen, die in *long-running activities* auftreten, und beschreiben eine Strategie, wie ebendiese im Deutschen genannten *langen Transaktionen* implementiert werden können. Sie beschreiben dabei beispielhaft Situationen, bei denen nach eingegangenen Aufträgen zuerst notwendige Genehmigungen verschiedener Akteure einzuholen sind, bevor der beauftragte Prozess abgeschlossen werden kann. Im Kern schlagen sie vor, solche Aktivitäten in mehrere Teil-Transaktionen aufzuteilen. Durch die Einführung von *Regeln* können demnach weitergehende Operationen ausgelöst werden, die verschiedene Fälle unvorhersehbarer Ereignisse während der Aktivität abfangen und verarbeiten können. Eine Regel setzt sich dabei aus einem *Event*, einer *Bedingung* und einer *Aktion* zusammen. Als Event können dabei unterschiedliche Aktionen herangezogen werden, die entweder im Speichersystem selbst stattfinden (z. B. eine Transaktion) oder aus externen Quellen ausgelöst werden (z. B. durch einen wiederkehrenden terminierten Trigger). Die Bedingung wird in Form einer Abfrage auf das Speichersystem umgesetzt und muss in einem booleschen *Wahr* resultieren, um die zugehörige Aktion auszulösen. Die abschließende Aktion kann wiederum Operationen auf dem Speicher selbst oder in angeschlossenen Systemen vornehmen.

2.7 Vorarbeiten zu objektbasierten Versionskontrollsystemen für Produktdatenmodelle des Bauwesens

Gemäß den erläuterten Konzepten zu Transaktionen wird deutlich, dass die etablierte Vorgehensweise der föderierten Disziplinmodelle in BIM-basierter Zusammenarbeit gut mit den beschriebenen BASE-Prinzipien in Verbindung gebracht werden kann. Bei der BIM-Zusammenarbeit sollte eine Transaktion letztendlich als die Synchronisierung von Informationen zwischen den am Austauschworkflow beteiligten Parteien verstanden werden. Weitergehend passen die in der Baubranche gegebenen Randbedingungen gut zu den zuvor erläuterten Überlegungen *langer Transaktionen* wie sie in Abschnitt 2.6 dargelegt wurden. Semenov und Jones (2015) begründen diese Überlegung im Detail und heben die damit verbundenen Vorteile hervor.

Neben den grundlegenden Entwicklungen zu Transaktionssystemen erfolgten in dieser Zeit auch diverse Weiterentwicklungen zum Umgang mit objektbasierten Repräsentationen und deren Versionierung. Wesentliche Aspekte wurden hierzu beispielsweise von Sciore (1994), Zeller und Snelting (1997) sowie Katz (1990) dokumentiert. Im Kontext geometrisch-semantischer Produktdatenmodelle für das Bauwesen sind aber insbesondere die Arbeiten

hervorzuheben, die zwischen 2000 und 2010 im Kontext des DFG-Schwerpunktprogramms 1103 *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau* entstanden sind.

Weise (2006) hat sich in seiner Dissertation dem Problem verteilt vorliegender Fachinformationen und der Gefahr inkonsistenter Kopien gewidmet und eine Methodik vorgeschlagen, die zu ändernde Teildatensätze aus einem Gesamtmodell herauslöst, modifizieren lässt und anschließend wieder in ein Gesamtmodell integriert. Genauer wurden diese einzelnen Schritte in vorangegangenen Veröffentlichungen erörtert. Weise et al. (2004) haben die Prinzipien zum Aus- und Einchecken partieller Modelländerungen in einer Client-Server-Umgebung dargelegt. Verschiedene Teile eines Gesamtmodells werden (nach Möglichkeit parallel) von verschiedenen Akteuren bearbeitet und sollen hierfür gemäß dem *Generalised Model Subset Definition Schema* (Weise et al., 2003) bearbeitet werden. Die vorgenommenen Bearbeitungen bilden in der Regel eine Sequenz mehrerer einzelner Modifikationen, die zu einem beliebigen Zeitpunkt wieder in das Gesamtmodell integriert werden sollen. Für das Re-Integrieren von Änderungen kritisieren die Autoren allerdings die bisher geringe Aufmerksamkeit, sodass entweder von idealen Bedingungen und strikt getrennten Teilmodellen ausgegangen wird oder ein hohes Maß an manuellen Eingaben und Selektionen notwendig ist. In der Folge beschreiben sie diverse Herausforderungen, die insbesondere bei der Verwendung des IFC-Datenmodells auftreten und beispielsweise auf die inkonsistente Vergabe von eindeutigen Merkmalen zurückzuführen sind. Darüber hinaus führen die Autoren auf, dass manche semantischen Änderungen weiterhin der Entscheidung von Endnutzern bedürfen. Insbesondere bei Änderungen, die implizite Auswirkungen auf andere Modellbestandteile haben können, sind weitergehende Nutzerentscheidungen während des Integrationsprozesses notwendig. Als Beispiel werden unter anderem die Änderung von Einheiten oder des Basiskoordinatensystems angeführt, die als Teilmodell beschrieben sind, aber nach deren Integration enorme Auswirkungen auf andere Bestandteile des Gesamtmodells haben können.

Koch und Firmenich (2011) haben einen umfassenden Ansatz für die Versionsverwaltung von prozeduralen Gebäudedarstellungen vorgestellt. Ihr Ansatz basiert darauf, sowohl Entwurfszustände als auch Zustandsübergänge darzustellen, wodurch der Austausch von informationsbasierten Änderungen mittels Entwurfsschritten ermöglicht wird, die als Modellierungsvorgänge bezeichnet werden. Hierfür setzen sie prozedurale Beschreibungen zur Erfassung der Modellierungsschritte ein, welche eine anschließende Versionierung der resultierenden Modelle ermöglichen. In der Veröffentlichung von Firmenich et al. (2005) wird zudem vorgeschlagen, die zu versionierenden Informationen aus dem BIM-Modell zu extrahieren und in einzelne XML-basierte Dokumente zu speichern. Diese können bei Bedarf auch wieder zu einem IFC-Modell zusammengesetzt werden. Die einzelnen Objektdokumente werden wiederum in textbasierten Versionskontrollsystemen verwaltet und beschreiben einzelne Objekte prozedural. Die gesamte Systemarchitektur beschreibt Richter (2009) und beleuchtet dabei insbesondere die gewählte Client-Server-Architektur sowie notwendige Befehle, die Endnutzern für die Interaktion mit dem System bereitgestellt werden müssen.

Etwa zehn Jahre später haben verschiedene Forschende die Grundideen erneut aufgegriffen. Postle schlägt in seiner *ifc-git* Implementierung [4] vor, das Versionskontrollsystem Git als Basis für die Übertragung inkrementeller Änderungen in IFC-Modellen zu nutzen. Dabei werden vorgenommene Änderungen auf Basis der serialisierten, textuellen SPF-Repräsentation ermittelt. Kritisch zu bewerten ist an diesem Ansatz, dass die erzeugenden Autorensysteme stets in gleicher Art und Weise die Instanzen exportieren. Eine Änderung der Exportreihenfolge oder die Vergabe neuer Entitätennummerierungen würden zu unbrauchbaren Diff-Ergebnissen führen. Zudem sieht der Vorschlag vor, dass die Entitätennummerierungen eines gelöschten Objekts nicht neu vergeben werden, um bei einer Umkehrung des Inkrements diese Zeilennummern für ebendieses Element reaktivieren zu können. Wenngleich dieser Ansatz als wichtiger Beitrag hinsichtlich der präzisen Überwachung von Modellinformationen zu sehen ist, schränken die getroffenen Annahmen die erfolgreiche Anwendung des Systems erheblich ein und und können dieses in der Praxis schnell zur Handlungsunfähigkeit bringen.

Poinet et al. (2020) stellt die Speckle-Plattform vor, die eine Versionskontrolle auf Basis einzelner Objekte im BIM-Modell bietet. Im Unterschied zu den Überlegungen von Firmenich, Nour und Koch verwendet die Speckle-Plattform JSON-Repräsentationen, die flexibel mit Informationen befüllt werden. Speckle zielt darauf ab, ein objektbasiertes Kollaborationssystem für BIM-Daten bereitzustellen, indem eine direkte Kommunikation zwischen Sendern und Empfängern implementiert wird. Der gesamte Datenaustausch basiert auf einer generischen Objektdefinition namens *SpeckleObject*, die dynamisch erstellte Attribute enthält [5]. Daher korreliert der Ansatz mit grundlegenden Konzepten von dynamisch typisierten Sprachen wie Python und ermöglicht es dem Benutzer, die Attribute eines bestimmten Objekts flexibel zu erstellen und zu ändern. Grundsätzlich erfordert die Verwendung des Speckle-Systems, dass alle angeschlossenen Systeme Speckle-Objektarchitektur in gleicher Weise formulieren und interpretieren. Infolgedessen erscheint der Ansatz vielversprechend, birgt aber auch das Risiko, einen weiteren Datenaustauschstandard in der Branche zu schaffen, der nur eine ausgewählte Reihe von Systemen verbindet.

Ergänzend zu diesen Arbeiten haben Schapke et al. (2021) die Vor- und Nachteile von optimistischer und pessimistischer Nebenläufigkeitskontrolle in modellbasierten Austauschszenarien aus einer globaleren Perspektive erörtert. Gemäß ihren Aussagen erfordert die Implementierung einer pessimistischen Nebenläufigkeitskontrolle oft Sperrmechanismen, um andere daran zu hindern, Informationen im Rahmen einer laufenden Transaktion zu manipulieren. Im Gegensatz dazu ermöglichen wie beschrieben optimistische Nebenläufigkeitssysteme eine höhere Flexibilität in entkoppelten, asynchronen Umgebungen.

Shi et al. (2018) haben zum Abgleich zweier BIM-Modelle im IFC-Datenmodell einen Algorithmus entwickelt, den sie als *IfcDiff* bezeichnen. Mit diesem ist es möglich, die

[4] https://github.com/brunopostle/ifc-git und https://blenderbim.org/docs/users/git_support.html (letzter Zugriff: 13.09.2023)

[5] Das *SpeckleObject* wurde inzwischen in *Base* umbenannt, implementiert aber weiterhin die beschriebenen Konzepte. Weitere Informationen sind der Speckle Dokumentation zu entnehmen: https://speckle.guide/dev/base.html (letzter Zugriff: 24.11.2023)

Unterschiede zwischen zwei IFC-Modellen durch Analyse der zugrundeliegenden Objektstrukturen zu ermitteln. Hierfür überführen sie die zwei zu vergleichenden IFC-Dateien in eine hierarchische Struktur und normalisieren sämtliche Informationen, die mehrfach im Modell vorhanden sind (beispielsweise alle Instanzen der Entität *IfcCartesianPoint*, die den Koordinatenursprung repräsentieren). Als Startpunkt der Traversierung werden dabei Instanzen gewählt, die keine ausgehenden Beziehungen besitzen (sog. *Terminal Nodes*). Stimmen zwei Instanzen aus den zu vergleichenden Modellen überein, werden anschließend alle Instanzen untersucht, die auf das übereinstimmende Paar verweisen. Auch unter diesen wird anschließend die Normalisierung durchgeführt und der Vergleich der Instanzen auf der nächsthöheren Ebene vollzogen. Durch Rekursion werden somit sukzessiv beide Modelle traversiert und übereinstimmende Paare in einer Liste gespeichert. Diese stellt nach Abschluss die Übereinstimmung der Modelle dar. Der Grad der Übereinstimmung zwischen den beiden Modellen wird abschließend durch die Division aller gemeinsamer Elementpaare durch die Anzahl der Instanzen je Modell definiert. Wenngleich die vorgenommene Normalisierung das dargelegte Vorgehen unterstützt, kann der vorgestellte Ansatz kritisch eingeordnet werden, da die Übereinstimmung lediglich durch einen mengenbasierten Vergleich der Instanzen erfolgt. Er gibt hingegen keinen Aufschluss über die tatsächlichen Änderungen, die im Modell vorgenommen wurden, und ist damit für weitere Betrachtungen im Sinne einer Versionskontrolle nur bedingt geeignet. Eine umfangreiche Metrik zum Quantifizieren von Ähnlichkeiten und Unterschieden zweier IFC-Modelle haben Lee et al. (2011) vorgestellt. Die vorgestellten Begriffe und Definitionen dienen zur Quantifizierung der Ähnlichkeiten und Unterschiede. Diese haben letztendlich das Ziel, die Übereinstimmung zwischen zwei IFC-Dateien zu quantifizieren, die in einem Datenaustauschprozess verwendet und erzeugt wurden. Wie auch im Ansatz von Shi et al. (2018) sind die ermittelten Kennwerte nur bedingt für die Nutzung im Versionsmanagement nützlich.

Auch im Kontext von Stadtmodellen existieren Vorarbeiten, die Änderungen zwischen zwei Modellversionen ermitteln. Hierbei sind insbesondere die Arbeiten von Nguyen und Kolbe (2020, 2022) zu erwähnen, in denen neben der Ermittlung der veränderten Modellinformationen auch diverse Interpretationen der Inkremente umgesetzt wurde.

Bei allen vorgestellten Ansätzen ist kritisch festzuhalten, dass die Versionskontrolle, der Informationsaustausch oder lediglich das Resultat aus dem Modellvergleich nicht auf den tatsächlichen Objektmodellen, sondern auf abgeleiteten Repräsentationen erfolgt. Dies birgt enormes Risiko der Fehl- oder Missinterpretation einzelner Informationen und kann zu einem potenziellen Informationsverlust führen. Vielversprechender sind daher Ansätze, die noch direkter auf den Objektmodellen interagieren und die gegebenen Informationen in einem Modell ungefiltert verarbeiten können. Auch hierzu existieren vereinzelt Vorarbeiten. Han et al. (2023) schlagen beispielsweise vor, die Instanzen eines IFC-Modells in ihrer vorliegenden (serialisierten) Form zu nutzen. Um allerdings schnell modifizierte Teile des Modells zu erkennen, werden Hashsummen eingesetzt, um einerseits Instanzen mit gleichen Informationen im Modell zu normalisieren und andererseits die Abhängigkeiten zwischen Instanzen mithilfe der ermittelten Hashsummen auszudrücken. Da die Entitäten-

nummerierung nicht mit in die Berechnung der Hashsumme eingeht, ermöglicht dieser Ansatz bereits eine gewisse Unabhängigkeit von der angewandten Serialisierungsreihenfolge. Der Ansatz baut damit auf den Überlegungen von Shi et al. (2018) auf, die ebenfalls die kontinuierliche Substitution der Entitätennummerierung in SPF-Serialisierungen mithilfe von Hashsummen evaluiert hatten.

2.8 Einordnung bestehender Ansätze zur Versionskontrolle

Die vorangegangenen Ausführungen zeigen, dass für die Beschreibung von Objektmodellen und in Teilen auch für deren Versionskontrolle bereits Ansätze existieren. Bedauerlicherweise zeigen alle bisherigen Systeme erhebliche Limitationen in ihrer Anwendbarkeit für die einheitliche Versionskontrolle unterschiedlicher Datenmodelle oder untersuchen die vorliegenden Informationen lediglich in ihrer serialisierten Form.

Zusammenfassend lassen für die Entwicklung von Versionskontrollmechanismen für Objektmodelle im Bauwesen die folgenden Beobachtungen festhalten:

- Es existieren umfangreiche Datenmodelle zur Beschreibung der gebauten Umwelt, die sich verschiedener Auszeichnungssprachen zur Serialisierung von Instanzen und zur Beschreibung des zugehörigen Datenmodells bedienen.
- Wenn ein Datenmodell für den Informationsaustausch ausgewählt wurde, können die serialisierten Darstellungen in verschiedenen Formen beschrieben werden.
- Mit der jeweiligen Beschreibungsform des Modells korreliert eine passende Basistechnologie zur Serialisierung von Objekten aus dem internen Speicher in eine zeichencodierte Schreibweise. Dabei müssen zur Abbildung von Relationen Identifkationsmerkmale für Objekte eingeführt werden, die arbiträr gewählt werden können.

Aus diesen Aspekten folgt, dass sich modifizierte MOF-M0 Instanzen in einer möglichst generalisierten Darstellung behandelt werden sollten, um Änderungen zwischen Versionsschritten möglichst allgemein und unabhängig von den genutzten Serialisierungstechnologien zu beschreiben. Entscheidend ist die Erkenntnis, dass selbst bei der Wahl der gleichen Auszeichnungssprache Repräsentationen auftreten können, die sich in ihrer textuellen Form unterscheiden, aber die gleiche ingenieurmäßige Information transportieren. Gegeben sei beispielhaft ein Datenmodell in Algorithmus 2.11, welches die Lage und Abmessungen von Rechtecken in einem kartesischen Koordinatensystem beschreibt. Definiert werden die beiden Klassen *Rechteck* und *Koordinate* in der EXPRESS Modellierungssprache nach ISO 10303-11.

Algorithmus 2.11: Einfaches Datenmodell zur Beschreibung von Form und Lage von Rechtecken innerhalb eines kartesischen Koordinatensystems

```
SCHEMA Geometrieklassen;

ENTITY Rechteck
    Name: STRING
    Breite: REAL
    Hoehe: REAL
    Mittelpunkt: Koordinate
END_ENTITY;

ENTITY Koordinate
    X: REAL
    Y: REAL
END_ENTITY;

END_SCHEMA;
```

Betrachtet werden soll nun die in Abb. 2.11 dargestellte Situation.

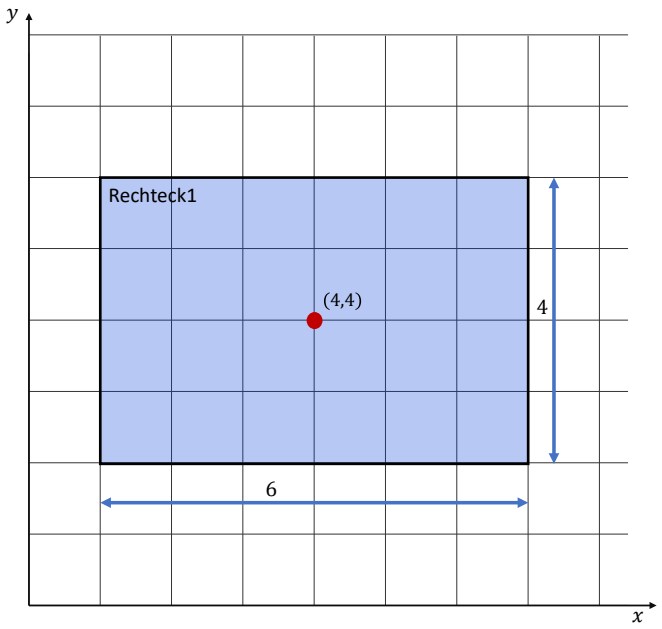


Abbildung 2.11: Beispiel einer Domäne, die mit dem in Algorithmus 2.11 definierten Datenmodell abstrahiert werden soll

Passende Instanzen des in Algorithmus 2.11 definierten Datenmodells stellen die SPF-Repräsentationen in Algorithmus 2.12 und Algorithmus 2.13 dar:

Algorithmus 2.12: Variante 1

```
ISO-10303-21;
HEADER;
FILE_SCHEMA(('Geometrieklassen'));
ENDSEC;

DATA;
#1=Rechteck('R1',6.0, 4.0, #2);
#2=Koordinate(4.0, 4.0);
ENDSEC;
END-ISO-10303-21;
```

Algorithmus 2.13: Variante 2

```
ISO-10303-21;
HEADER;
FILE_SCHEMA(('Geometrieklassen'));
ENDSEC;

DATA;
#1=Koordinate(4.0, 4.0);
#2=Rechteck('R1', 6.0, 4.0, #1);
ENDSEC;
END-ISO-10303-21;
```

Vergleicht man die beiden serialisierten Varianten nun mit einem textuellen Diff (wie in Abschnitt 2.5.2 erklärt), ergibt sich das in Algorithmus 2.14 dargestellte Resultat:

Algorithmus 2.14: Diff-Operation zwischen den in Algorithmus 2.12 und Algorithmus 2.13 dargestellten Textsequenzen

```
$ diff Version1.spf Version2.spf
7,8c7,8
< #1=Rechteck('R1',6.0, 4.0, #2);
< #2=Koordinate(4.0, 4.0);
---
> #1=Koordinate(4.0, 4.0);
> #2=Rechteck('R1', 6.0, 4.0, #1);
```

Erkennbar ist, dass der Algorithmus die Zeilen 7 und 8 als gelöscht und neu hinzugefügt identifiziert. Trotzdem beschreiben beide Repräsentationen dieselbe Domäne, allerdings sind die beiden Instanzen in unterschiedlicher Reihenfolge serialisiert worden. Aus diesem einfachen Beispiel geht anschaulich hervor, dass eine Versionskontrolle von Informationen in den derzeit üblichen Auszeichnungssprachen und mit den aktuell verfügbaren Systemen keinen sinnvollen Beitrag zur Verbesserung der Probleme mit sich bringt oder umfangreiche Annahmen getroffen werden müssen.

Gesucht wird demnach eine Methodik zur Versionsüberwachung, die die in einem Modell gespeicherten Informationen im Sinne der Grundprinzipien objektorientierter Modellierung, aber unabhängig von der konkreten Semantik behandelt. Naheliegend ist daher, BIM-Modelle nicht in ihrer serialisierten Form zu betrachten, sondern die Überlegungen auf die zugrundeliegenden, vernetzten Objektstrukturen zu lenken. Konzepte der Graphtheorie und der Graphtransformation eignen sich sehr gut, um solche Netzwerke mit allgemeinen Begriffen der Mathematik zu modellieren (Heckel & Taentzer, 2020). Für deren Umformung werden ergänzend Grundlagen der Graphersetzung beziehungsweise der Graphtransformation erläutert.

Kapitel 3

Graphen und graphbasierte Darstellungen von Objektnetzwerken

Aus den in Kapitel 2 zusammengetragenen Erkenntnissen ergibt sich, dass alphanumerischer Serialisierungsverfahren zwar strukturierte Informationen übertragen können, aber sich nur begrenzt für die Analyse und Interaktion mit den zugrundeliegenden semantischen und topologischen Abhängigkeiten zwischen den eigentlichen Objektbeschreibungen eignen. Aufgezeigt wurden insbesondere Abhängigkeiten von der gewählten Serialisierungsreihenfolge, aber auch die Abhängigkeit von verschiedenen Kodierungsformen stellt eine nicht unwesentliche Limitation für die Analyse der zugrundeliegenden Graphstrukturen dar.

Folglich wird im weiteren Verlauf auf wesentliche Grundlagen der aus der Mathematik bekannten *Graphentheorie* eingegangen. Graphen ermöglichen als allgemeines mathematisches Konzept die Beschreibung von Objekten und deren Beziehungen zu einander. Sie erweitern damit den einfachen *Mengenbegriff* um die Möglichkeit, Zusammenhänge zwischen Elementen formal zu beschreiben. Dafür werden nun ausgewählte Grundlagen der Graphentheorie und Begrifflichkeiten eingeführt, die im weiteren Verlauf zur Beschreibung vernetzter Informationen in Modellen herangezogen werden. Die verwendeten Begriffe orientieren sich vor allem an den Veröffentlichungen von Diestel (2017) sowie Bollobás (1998). Die stringente Abfolge der einzelnen Begriffe ist ebenfalls von diesen Publikationen inspiriert und greift zudem Aspekte auf, die Vilgertshofer (2022) erläutert hat. An manchen Stellen werden in zitierten Veröffentlichungen Synonyme für bestimmte Konzepte verwendet, die möglichst harmonisiert verwendet wurden, um die durchgängige Verständlichkeit zu erhöhen.

3.1 Theoretische Grundbegriffe

3.1.1 Definitionen

Abstrakt wird ein Graph G durch die Mengen von Knoten V und Kanten E beschrieben:

$$G = (V, E) \tag{3.1}$$

Ein Knoten $v \in V$ repräsentiert ein einzelnes, eindeutig identifizierbares Objekt in der Gesamtmenge aller Knoten V, wohingegen eine Kante $e \in E$ eine Beziehung zwischen zwei Objekten aus der Knotenmenge V abbildet. Die Kantenmenge E ist dabei eine zweiele-

mentrige Teilmenge der Knotenmenge V: $E \subseteq [V]^2$ (Diestel, 2017). Kanten können dabei gerichtet oder ungerichtet sein. In Formelausdrücken wird eine Kante zwischen den Knoten u und v entweder mit $[e, f]$, mit einem Pfeil $e \to f$ oder verkürzt mit ef notiert. Sofern nichts anderes benannt wird, werden allerdings gerichtete Kanten angenommen, sodass das erste Tupelelement den Startknoten und das zweite Tupelelement den Zielknoten der Kante bezeichnet.

Als *schlichter* Graph wird eingrenzend ein Graph verstanden, der sich aus einer endlichen, nicht leeren Knotenmenge V mit p Knoten und einer bestimmten Kantenmenge E zusammensetzt. Betrachtet man zwei Knoten u und v aus V, so bezeichnet *Adjazenz* den Zustand, wenn die Knoten u und v durch eine Kante verbunden sind. Man sagt auch, dass der Knoten u *adjazent* zu v ist. Weiter bezeichnet man die Knoten u und v als *inzident* zu einer Kante e, wenn diese die Begrenzungen der Kante e definieren. Zwei Kanten e und f werden als *benachbart* bezeichnet, wenn sie einen gemeinsamen Knoten u referenzieren. Die Folge mehrerer benachbarter Kanten bezeichnet man auch als *Pfad* oder *Weg*. Als Synonyme zu dem Begriff des Pfads werden in anderen Publikationen auch die Begriffe des Weges oder des Kantenzugs beziehungsweise der Kantenfolge herangezogen. Der Begriff *Schleife* bezeichnet eine Kante, die einen Knoten mit sich selbst verbindet. Darüber hinaus nennt man einen Pfad *Zyklus*, bei dem der Start- und Endknoten identisch sind.

Neben schlichten Graphen gibt es weitere Formen, die in verschiedenen Bereichen Verwendung finden. *Hypergraphen* verfügen beispielsweise über die Möglichkeit, mehr als zwei Knoten durch eine Kante zu verbinden. Der Definitionsraum der Kantenelemente ist somit nicht mehr auf ein zweidimensionales Tupel beschränkt, sondern kann prinzipiell beliebig viele Dimensionen einnehmen (Bauderon & Jacquet, 1997). Der Begriff des *Multigraphs* hingegen beschreibt einen Graph, bei dem eine Kante $e = (u, v)$ zwischen zwei Knoten mehr als einmal in der Kantenmenge auftritt. Damit muss mathematisch gesehen die Kantenmenge nicht mehr als einfache Menge, sondern als sogenannte *Multimenge* betrachtet werden.

Als *benannter* oder *gelabelter* Graph wird eine Struktur verstanden, bei der Knoten und Kanten alphanumerisch beschriftet werden. Dies kann beispielsweise numerisch oder mit Buchstaben erfolgen. Eine Beschriftung kann grundsätzlich mehrfach vergeben werden. Vereinfachend werden in diesem Abschnitt aber vorerst eindeutige Kennzeichen verwendet. Abb. 3.1 illustriert ein kompaktes Beispiel, eines benannten, schlichten Graphs mit gerichteten Kanten.

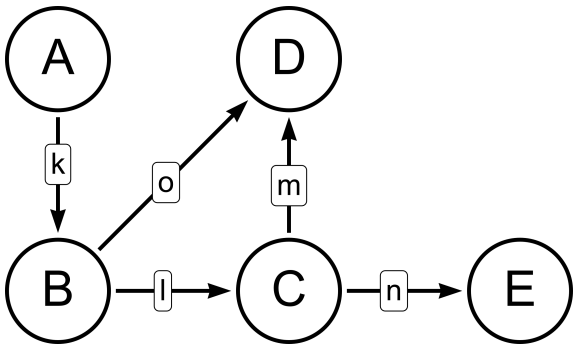


Abbildung 3.1: Schlichter Graph G

Ein Graph G' ist ein *Teilgraph* eines anderen Graphs G, wenn V' und E' jeweils Teilmengen der Knotenmenge V und Kantenmenge E sind:

$$V' \subseteq V$$

$$E' \subseteq E$$

$$G' \subseteq G$$

Die Begriffe *Subgraph* oder *partieller Graph* werden synonym für den Ausdruck Teilgraph verwendet. Als besonderer Fall existiert der *induzierte* Teilgraph. Dabei handelt es sich um einen Teilgraph, der alle zu den Knoten benachbarten Kanten im Teilgraph enthält. Abb. 3.2 zeigt den zuvor eingeführten schichten Graph G, einen Teilgraph G' mit den Knoten A, B und C und der Kante l sowie den induzierten Teilgraph G'', der zusätzlich die Kanten m und o enthält.

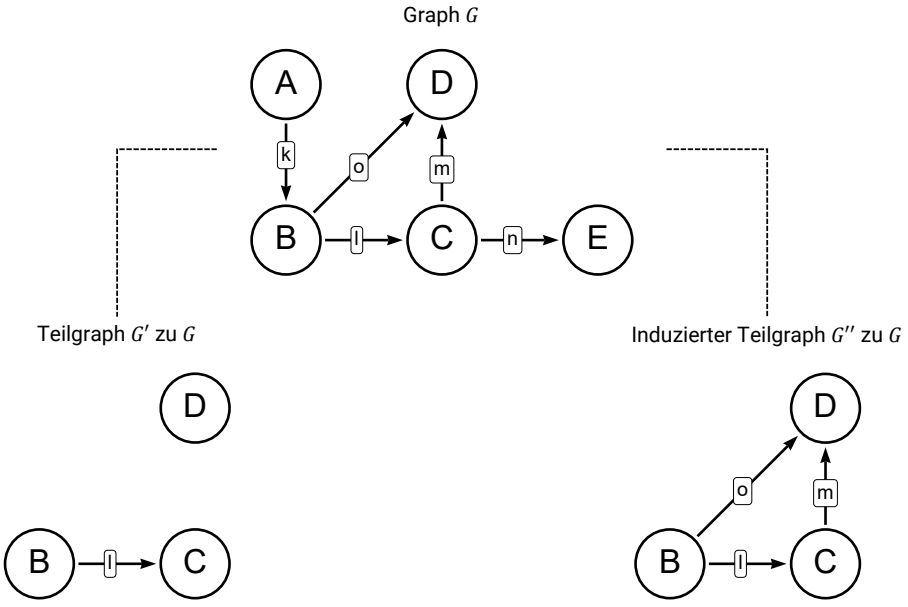
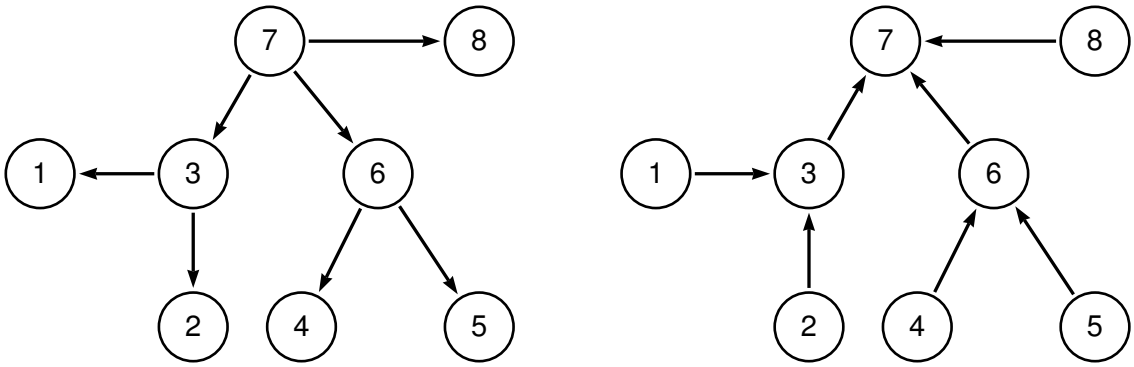


Abbildung 3.2: Teilgraph G' und induzierter Teilgraph G'' zu dem in Abb. 3.1 eingeführten Graph G

Ergänzend zu dem Begriff des *Teilgraphs* bezeichnet eine *Clique* einen Teilgraph $G' \subseteq G$, bei der G' vollständig ist. Als *vollständig* wird wiederum ein Graph bezeichnet, wenn alle Knoten $v \in V$ miteinander verbunden sind.

Spezielle Eigenschaften weist ein sogenannter *Baum* auf. Mit diesem Begriff wird ein Graph bezeichnet, der zusammenhängend ist und keine geschlossenen Pfade enthält. Folglich existiert zwischen zwei Knoten des Graphs immer exakt ein Pfad, der diese Knoten verbindet. Bemerkenswert ist zudem, dass jeder induzierte Teilgraph eines *Baums* erneut ein *Baum* ist und dieser durch exakt eine Kante mit dem übrigen Graph verbunden ist. Diese Eigenschaft wurde bereits kurz in ihrer Anwendung für die hierarchischen Auszeichnungssprachen XML und JSON und deren inkrementelle Verfahren genutzt, um die Lage von Modifikationen in diesen Dokumenten durch einen entsprechenden Pfad zu beschreiben.

Kanten eines *Baums* können gerichtet oder ungerichtet sein. Sofern gerichtete Kanten verwendet werden, kann weiter zwischen *Out-Trees* und *In-Trees* unterschieden werden. Bei Out-Tress existiert exakt ein Knoten $v \in V$ ohne eingehender Kante, der als Wurzelknoten bezeichnet wird. Alle anderen Knoten verfügen über exakt eine eingehende Kante sowie keiner, einer oder mehrerer ausgehender Kanten. Sofern ein Knoten keine ausgehende Kante besitzt, wird er auch als Blattknoten bezeichnet. Bei In-Trees gelten die Überlegungen invers. Abb. 3.3a zeigt ein Beispiel eines Out-Trees, wohingegen Abb. 3.3b die Kanten umgekehrt wurden und es sich damit um einen In-Tree handelt. Der Wurzelknoten ist jeweils mit 7 beschriftet.



(a) Beispiel eines Out-Trees (b) Beispiel eines In-Trees

Abbildung 3.3: Beispiele für Bäume

Homomorphismus und Isomorphismus

Für viele Fragestellungen, für die graphbasierte Beschreibungen eingesetzt werden, ist es relevant, ob eine Abbildungsvorschrift zwischen zwei Graphen G und H existiert. In Gleichung (3.2) beschreibt φ die *Abbildung*, bei der jedes Element aus einer Definitionsmenge D einem Element der Ergebnismenge Z zugeordnet wird.

$$\varphi : D \to Z \tag{3.2}$$

Damit lassen sich die Begriffe der Bijektivität, Injektivität und Surjektivität einführen. Eine bijektive Abbildung bildet jedes Element der Definitionsmenge auf exakt ein Element der Ergebnismenge ab. Als injektive Abbildung bezeichnet man weitergehend einen Fall, bei der die Zielmenge weitere Elemente enthält, die durch die Abbildung der Definitionsmenge auf die Zielmenge nicht erreicht werden. Die Zielmenge muss daher mindestens die gleiche Mächtigkeit wie die Definitionsmenge aufweisen. Den gegenteiligen Sachverhalt beschreibt schließlich eine surjektive Menge, bei der mehr als ein Element der Definitionsmenge auf ein Element der Zielmenge abgebildet wird. Abb. 3.4 veranschaulicht die Begriffe für die Definitionsmenge $D = \{1; 2; 3; 4\}$ und die Ergebnismenge $Z = \{A; B; C; D\}$.

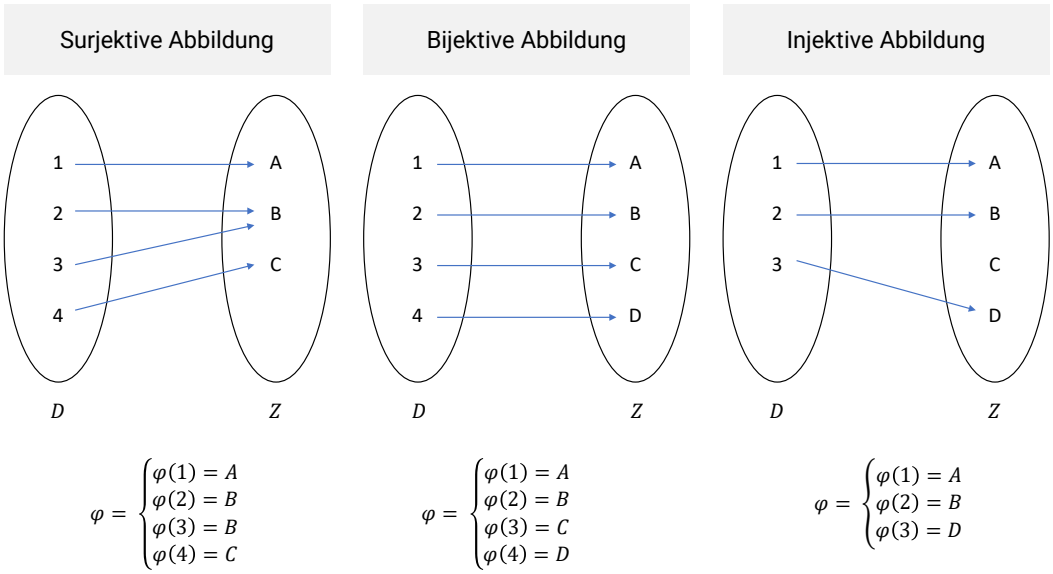
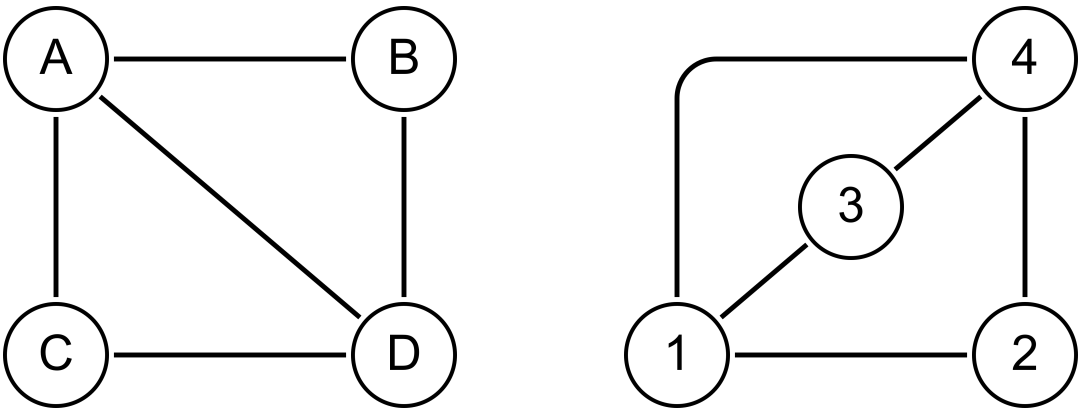


Abbildung 3.4: Abbildungsarten zwischen einer Definitionsmenge D und einer Ergebnismenge Z

Abbildungen können nun auch zwischen zwei Graphen G und H betrachtet werden. Man bezeichnet G als *homomorph* zu H, wenn eine Abbildung φ zwischen G und H existiert, die die Nachbarschaften zwischen den Knoten erhält, also jede Kante aus G auf eine Kante in H abgebildet wird. Damit werden demnach alle Nachbarschaften zwischen den Knoten in G und H erhalten. Formal gilt demnach:

$$(x, y) \in G \rightarrow (\varphi(x), \varphi(y)) \in H \tag{3.3}$$

Weiter beschreibt ein *Isomorphismus* die Abbildung zwischen zwei Graphen G und H, die *homomorph* sind und deren Knotenabbildung $\varphi(V)$ zusätzlich bijektiv ist. Die zwei Graphen G und H besitzen folglich dieselbe Anzahl von Kanten, Knoten sowie dieselben Inzidenzen der Kanten. Außerdem wird jeder Knoten aus dem Graph G exakt einem Knoten aus dem Graph H zugeordnet. Für die Veranschaulichung dieser besonderen Abbildungen werden in der Folge die ungerichteten Graphen G und H betrachtet, die in Abb. 3.5 dargestellt sind.



(a) Graph G (b) Graph H

Abbildung 3.5: Isomorphismusbetrachtungen zwischen zwei Graphen G und H

Zwischen den Graphen G und H kann folgende Abbildung der Knotenmengen notiert werden:

$$\varphi = \begin{cases} \varphi(A) \to 1 \\ \varphi(B) \to 3 \\ \varphi(C) \to 2 \\ \varphi(D) \to 4 \end{cases} \tag{3.4}$$

Damit folgt für die Abbildung der Kanten:

$$\begin{aligned} A \leftrightarrow B &\to 1 \leftrightarrow 3 \\ A \leftrightarrow C &\to 1 \leftrightarrow 2 \\ A \leftrightarrow D &\to 1 \leftrightarrow 4 \\ B \leftrightarrow D &\to 3 \leftrightarrow 4 \\ C \leftrightarrow D &\to 2 \leftrightarrow 4 \end{aligned} \tag{3.5}$$

Aufgrund der Bijektivität der Gleichung (3.4) sind die Graphen G und H isomorph zueinander.

Gleichzeitig könnte aber auch die folgende Abbildung φ' formuliert werden, die ebenfalls den isomorphen Charakter zwischen G und H aufzeigt:

$$\varphi' = \begin{cases} \varphi(A) \to 1 \\ \varphi(B) \to \mathbf{2} \\ \varphi(C) \to \mathbf{3} \\ \varphi(D) \to 4 \end{cases} \tag{3.6}$$

Getauscht wurden die Abbildungen der Knoten B und C. Die Abbildung der Kanten muss gleichermaßen angepasst werden. Gleichwohl könnten auch noch die Abbildungen von A und D auf die Knoten 1 und 4 getauscht werden, ohne das Ergebnis zu verändern. Insgesamt bleibt die Aussage über die isomorphe Abbildung zwischen G und H allerdings

unverändert. Festzuhalten ist allerdings, dass die Morphismus-Abbildungen zwischen zwei Graphen teilweise verschiedene Formen einnehmen können.

Bisher existiert kein effizienter Algorithmus, der eine isomorphe Abbildungsvorschrift aus zwei gegebenen Graphen effizient ermitteln kann. Das Problem gilt in der Informatik als NP-komplett (Bunke, 1997; Conte et al., 2007). Es existieren allerdings diverse Ansätze, die den größten gemeinsamen Teilgraph zwischen zwei Graphen G und H ermitteln können. Diese bedienen sich dabei häufig rekursiven Traversierungsverfahren, in der ausgehend von einem gemeinsamen Startpunkt beide Graphen G und H traversiert werden. Andere Verfahren untersuchen hingegen auftretende *Cliquen* und stellen zwischen diesen Relationen her. Grundsätzlich können in diesem Gebiet exakte Verfahren (Bunke et al., 2002) und approximative Methoden (Wang & Maple, 2005) unterschieden werden. Letztere spielen insbesondere in Ansätzen der künstlichen Intelligenz eine wesentliche Rolle, werden hier aber nicht weiter verfolgt.

Mustersuche und Passung

Als *Muster* (im Englischen *Pattern*) p wird eine Struktur bezeichnet, die in einem gegebenen Arbeitsgraph G gesucht werden soll. Das Muster p selbst ist dabei ebenfalls ein Graph und wird durch Knoten und Kanten ausgedrückt. Die aufgefundenen Teilgraphen $G' \subseteq G$ werden *Passung* genannt und stellen das Ergebnis der *Mustersuche* dar. Diese Teilgraphen sind isomorph zum Arbeitsgraph G. Für ein Muster p können keine, eine oder mehrere Passungen im Arbeitsgraph G gefunden werden. Eine einfache Form eines *Musters* stellt der *Weg* oder *Pfad* dar. Ein Weg kennzeichnet sich dabei durch die Suche nach einer Kantenfolge aus, wobei keine Schleifen oder Verzweigungen im Muster auftreten.

3.1.2 Graphtransformationen und Graphersetzung

Das zuvor beschriebene Suchen nach Mustern in einem Graph kann einerseits dazu genutzt werden, bestimmte Informationen mit unterschiedlicher Komplexität aus einer Graphstruktur abzufragen. Gleichzeitig bildet das Auffinden von Mustern aber auch die Grundlage für die Graphersetzung, dessen grundsätzliches Ziel die regelbasierte Modifikation eines Graphen durch Anwendung verschiedener deskriptiver Regeln ist. Den Ursprung der Graphersetzung findet man in der Literatur vor allem aufgrund vorgefundener Limitationen klassischer Ansätze zur Umformung von sequenziellen Textinformationen (Pfaltz & Rosenfeld, 1970). Insbesondere Sachverhalten, die mit beliebig vernetzte (Daten-)Strukturen repräsentiert werden müssen, waren frühere Verfahren der einfachen Textersetzung nicht gewachsen (Kniemeyer, 2008). Anwendung fand und findet die Graphersetzung heutzutage in verschiedenen Bereichen. Hervorzuheben sind besonders die Initiativen rund um die Optimierung im Übersetzerbau (Geiß, 2007) oder verschiedener Optimierungs- und Syntheseprobleme (Helms & Shea, 2012).

Wie bereits in Abschnitt 2.5.2 dargestellt, verfolgen inkrementelle Verfahren für Textsequenzen den Ansatz, Änderungen zwischen zwei Textsequenzen durch eine Sammlung von

Änderungsoperationen darzustellen. Diesen Grundsatz übernehmen Systeme der Graphtransformation, betrachten dabei aber nicht einfache Sequenzen (aus Buchstaben oder Zeilen), sondern die Gesamtheit eines Graphs aus seinen Knoten- und Kantenmengen. Zusätzlich sind Graphersetzungssysteme nicht nur dafür gedacht, eine Ersetzungsregel exakt einmal anzuwenden, um den Zustand einer Textsequenz von einer Version in eine andere zu verändern, sondern generalisieren die Anwendung einer Regel, um beispielsweise verschiedene Varianten zu ermitteln. Werden Ersetzungsregeln mehrfach angewandt, um eine Vielzahl verschiedener Ergebnisse zu erzielen, so spricht man auch von einer *Graph-Grammatik* (Kolbeck et al., 2022). Eine Graph-Grammatik besteht dabei aus einem Startzustand, der Sammlung an Transformationsregeln und einem Set an Abbruchkriterien. Ausgehend von diesen Komponenten werden alle möglichen Zustände generiert und die gefundenen Strukturen anschließend gegen definierte Kriterien evaluiert.

Eine Graphtransformation beschreibt formal die Modifikation eines Graphs von seiner Ausgangsstruktur G in eine Zielstruktur G' durch die Anwendung einer Transformationsregel p (Dörr, 1995; Habel et al., 2001; Heckel, 2006). Hierfür werden die folgenden Schritte ausgeführt:

Auffinden des spezifizierten Mustergraphs L

Im ersten Schritt muss der spezifizierte Mustergraph L (auch als *Precondition* bezeichnet) im Arbeitsgraph G gefunden werden. Formal wird ein Subgraph in $g \subseteq G$ gesucht, der homomorph zum Mustergraph L ist und damit eine Passung zum Mustergraph L darstellt. Mit diesem Morphismus werden gleichzeitig alle anderen Knoten und Kanten des Arbeitsgraphs geschützt, die durch die Ersetzungsregel nicht beeinflusst werden dürfen.

Löschen des Mustergraphs L und Einfügen des Ersetzungsgraphs R

Ziel dieses Schrittes ist es, die entsprechende Stelle im Arbeitsgraph so zu transformieren, dass dort ein isomorphes Abbild des Ersetzungsgraphs R entsteht. Hierfür wird zuerst der zu L homomorphe Subgraph aus dem Arbeitsgraph G entfernt. Dabei wird in Kauf genommen, dass kurzfristig sogenannte *hängende Kanten* (im Englischen *dangling edges*) auftreten, also Kanten, die nur nur noch über einen Knoten verfügen. Im Anschluss wird der Ersetzungsgraph R (auch als *Postcondition* bekannt) eingesetzt und durch die *dangling edges* mit dem Arbeitsgraph verknüpft. Es entsteht so der transformierte Graph G. Da die *dangling edges* die grundlegende Definition der Kantenmenge schlichter Graphen verletzen, kann hier für eine bessere Darstellung der Erhaltungsmorphismus r genutzt werden, der sich zwischen Mustergraph L und Ersetzungsgraph R einstellt. Anstatt den gesamten Mustergraph zu löschen, reicht es aus, wenn nur jene Knoten und Kanten im Arbeitsgraph G gelöscht werden, die nur im Mustergraph L, aber nicht im Ersetzungsgraph R enthalten sind. Gleichzeitig müssen lediglich jene Knoten und Kanten neu eingefügt werden, die nur in R aber nicht in L enthalten sind.

Abb. 3.6 zeigt ein Beispiel einer einfachen Graphersetzungsregel und ist von den Ausführungen in Dörr (1995) inspiriert. Gegeben sei der Graph G mit den benannten Knoten $\{a, b, c, d, e, f\}$ sowie der gerichteten Kantenmenge $\{[ab], [bc], [bd], [de], [df], [fe]\}$. Die

Graphersetzungsregel besteht wie beschrieben aus einem Mustergraph und einem Ersetzungsgraph. Während in der Schrittabfolge auf der linken Seite in Schritt (2) eine Passung des Mustergraphs in G gefunden wird, diese anschließend in (3) gelöscht wird und in (4) der Ersetzungsgraph eingesetzt wird, bedient sich die Darstellung auf der rechten Seite dem Erhaltungsmorphismus. Dieser Morphismus beschreibt einen Teilgraph, der sowohl zum Mustergraph als auch zum Ersetzungsgraph isomorph ist. Aus praktischer Sicht handelt es sich dabei um einen Teilgraph, der in beiden Regelbestandteilen enthalten ist und demnach erst gelöscht und später wieder eingefügt wird. Im gewählten Beispiel würde nach dem Löschen des Mustergraphs aus G eine hängende Kante entstehen, die zuvor die Knoten $[bd]$ verbunden hat. Da der Knoten b nach dem Löschen nicht mehr vorhanden ist, wird hier formal die getroffene Kanten-Definition eines schlichten Graphs verletzt. Betrachtet man allerdings den Erhaltungsmorphismus zwischen den beiden Regelbestandteilen, so wird ersichtlich, dass die Knoten a und b sowie die gerichtete Kante $[ab]$ sowohl in dem Muster- als auch im Ersetzungsgraph vorhanden sind. Demnach ist deren Löschung und anschließendes Wieder-Einfügen nicht notwendig. Gelöscht werden müssen lediglich jene Bestandteile des Mustergraphs, die nicht Teil des Erhaltungsisomorphismus sind. Im Beispiel sind dies der Knoten c sowie die Kante bc. Das gleiche Prinzip greift anschließend auch bei dem Einfügen des Ersetzungsgraphs. Hier folgt, dass lediglich der Knoten g sowie die Kanten ag und bg neu hinzuzufügen sind.

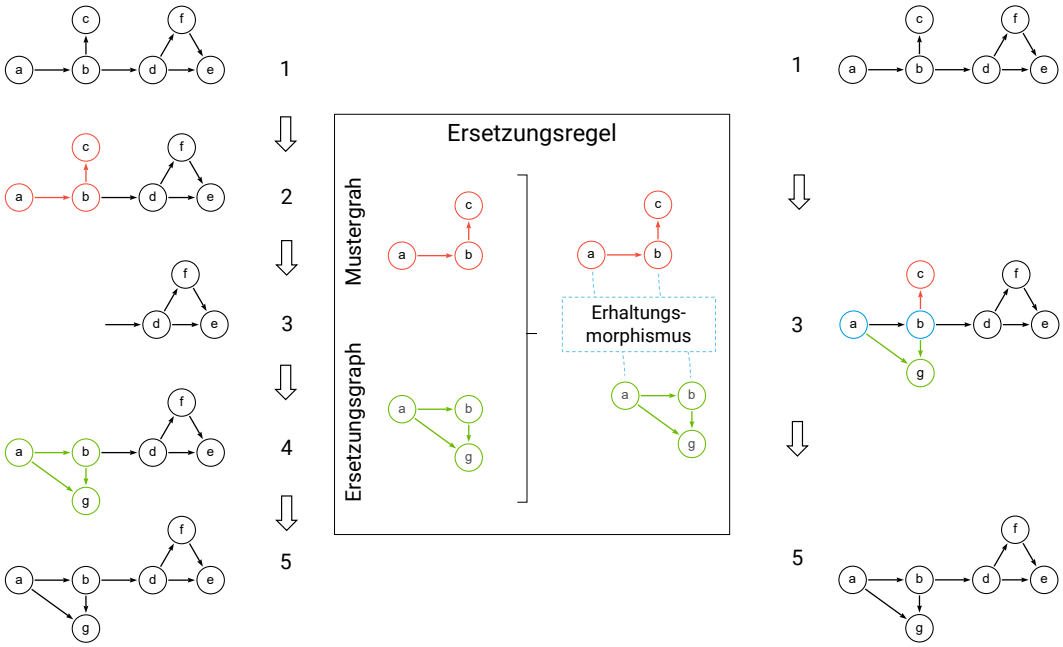


Abbildung 3.6: Gegenüberstellung der klassischen Graphtransformation mit Löschen des Mustergraphs und Einfügen des Ersetzungsgraphs sowie der Graphmodifikation unter Berücksichtigung des Erhaltungsmorphismus nach dem Single Push Out Verfahren

Das erläuterte Verfahren unter Berücksichtigung des Erhaltungsmorphismus ist in der einschlägigen Literatur auch als Single Push Out (SPO) bekannt. Als alternatives Verfahren existiert zusätzlich der sogenannte Double Push Out (DPO). Im Gegensatz zum SPO beschreibt der DPO die notwendigen Transformationen in zwei Stufen. Dabei wird der vorübergehende Zustand nach dem Entfernen und vor dem Einfügen explizit als Interface

I beschrieben. Dafür werden in der Graphtransformationsregel weitere Erhaltungsmorphismen spezifiziert. So beschreibt die isomorphe Passung des Mustergraphs L auf den zu verändernden Arbeitsgraph G den Startpunkt der Transformation und entspricht damit dem Auffinden des Mustergraphs analog zur Vorgehen im SPO (Precondition). Dieses Muster wird auch als *Left-Hand-Side* bezeichnet. Weitergehend wird nun der Zwischengraph I hinzugefügt, welche manchmal auch als *Interface* bezeichnet wird. I ist in seiner Beschaffenheit ein isomorpher Teilgraph zum Mustergraph L, was mit der Abbildungsvorschrift l beschrieben wird. Wie zuvor müssen alle Knoten und Kanten, die in I, aber nicht in L enthalten sind, aus dem Arbeitsgraph G entfernt werden. Nach dem Entfernen dieser Graphbestandteile ergibt sich ein Zwischenzustand, der als Kontextgraph D bezeichnet wird. Auch hierzu lässt sich eine isomorphe Abbildung zwischen I und D formulieren, die im Diagramm mit d bezeichnet wird. Der Kontextgraph D repräsentiert demnach den Zustand nach dem Löschen aller Knoten und Kanten aus G, die nicht in der Abbildung l zwischen I und L enthalten sind. Weiter wird in der Ersetzungsregel der Ersetzungsgraph R definiert, der auch *Right-Hand-Side* genannt wird. Wie zuvor existiert zwischen dem Interface I und dem Ersetzungsgraph R eine isomporphe Abbildung, sodass hier nach dem gleichen Prinzip jene Knoten und Kanten identifiziert werden können, die in den Kontextgraph D einzufügen sind, um den gewünschten Zielgraph G' zu erhalten (analog zur Postcondition im SPO). Zwischen dem Ersetzungsgraph R und dem Zielgraph G' besteht ebenfalls eine isomorphe Abbildung, die mit $m*$ bezeichnet wird.

Abb. 3.7 zeigt die zuvor behandelte Transformation als DPO-Konstruktion.

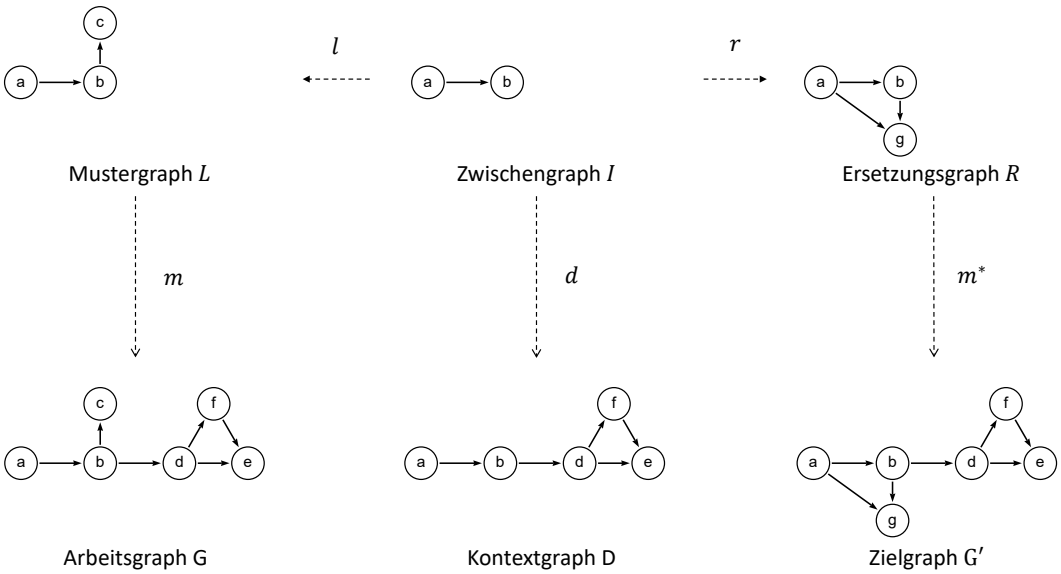


Abbildung 3.7: Formulierung einer Graphersetzungsregel als Double Push Out

3.1.3 Breiten- und Tiefensuche auf gerichteten Graphen

Neben der Transformation von Graphstrukturen bilden verschiedene Strategien zur Traversierung von Graphen die Grundlage für unterschiedliche Algorithmen, die beispielsweise

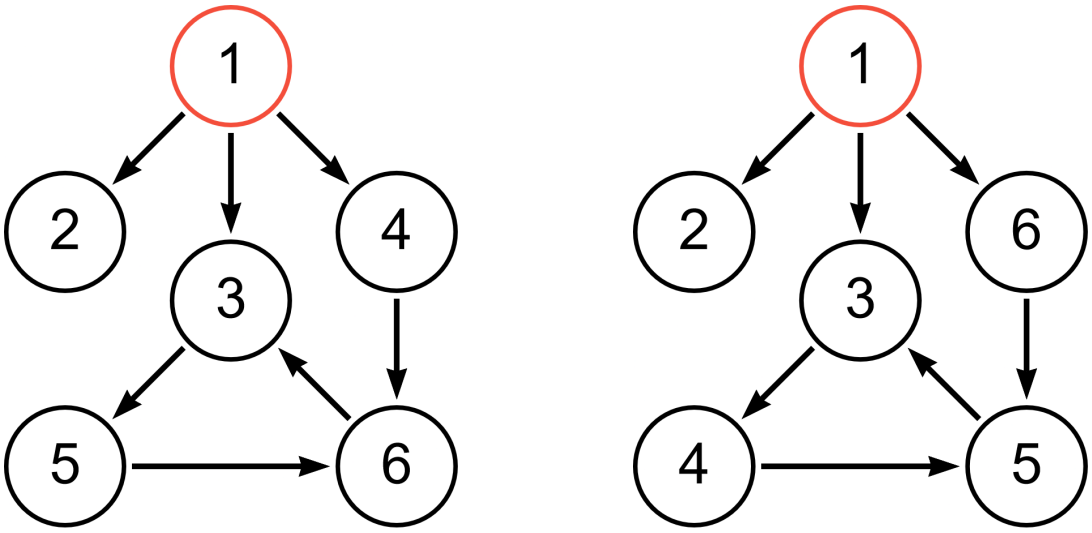
kürzeste Pfade oder grundsätzliche Erreichbarkeiten zwischen zwei Knoten ermitteln können. Die beiden wichtigsten Vertreter sind dabei die Breiten- und die Tiefensuche, die im Folgenden angelehnt an die Ausführungen von Cormen et al. (2009, S. 594–609) kurz eingeführt werden. Als Grundproblem wird in beiden Fällen angenommen, dass von einem Ausgangsknoten s ein Zielknoten w gefunden werden soll. Häufig werden die Algorithmen allerdings ohne konkreten Zielknoten genutzt, um die Knoten eines Graphs hierarchisch zu sortieren.

Gegeben sei ein schlichter Graph $G = (V, E)$, der den zuvor eingeführten Prinzipien folgt. Zu Beginn wird ein Speicher A initialisiert, der eine Liste von Knoten beinhalten kann und leer ist. Die Breitensuche startet an einem Startknoten s, der sich in der Regel aus dem Kontext des zu untersuchenden Problems ergibt. Dieser wird initial dem Speicher A hinzugefügt. Ausgehend vom Startknoten s untersucht der Algorithmus alle zu s adjazenten Knoten $v \in V$. Sofern ein Maß zur Berechnung der Entfernung zwischen s und jedem erreichbaren Knoten v ermittelt werden kann, wird diese zur Sortierung der Zielknoten auf der aktuellen Ebene genutzt und alle Knoten v werden in den Speicher A in aufsteigender Reihenfolge hinzugefügt. Sofern kein Merkmal für die Entfernung existiert, wird die Entfernung zu 1 angenommen und die Knoten werden ebenfalls in den Speicher hinzugefügt. Anschließend wird der aktuell erste Eintrag aus A als neuer Startknoten s gesetzt, aus A entfernt und wiederum alle direkt adjazenten Knoten untersucht. Um zu verhindern, dass die Suche in eine Endlosschleife gerät, wird zusätzlich an jedem Knoten vermerkt, ob von diesem bereits alle direkt adjazenten Knoten untersucht wurden. Damit wird unterbunden, dass bereits untersuchte Knoten erneut dem Speicher A zur weiteren Betrachtung hinzugefügt werden. Es handelt sich aufgrund des repetitiven Vorgehens daher um eine rekursive Funktion. Die Suche wird abgebrochen, sobald der gewünschte Knoten gefunden wurde oder alle Knoten entsprechend besucht wurden. Mit diesem Vorgehen lässt sich ein sogenannter *Breitensuchbaum* konstruieren, der die Reihenfolge der besuchten Knoten beinhaltet. Sei nun der kürzeste Pfad zwischen zwei Knoten s und w gesucht, so reduziert sich das Problem auf die Suche nach dem kürzesten Pfad im Breitensuchbaum.

Das Vorgehen der Tiefensuche ist jenem der Breitensuche sehr ähnlich. Im Gegensatz zur Untersuchung aller Knoten auf einer bestimmten Ebene wird bei der Tiefensuche zuerst möglichst tief in die Graphstruktur traversiert, bevor etwaige benachbarte Knoten auf einer bestimmten Ebene untersucht werden. Die Tiefensuche erkundet demnach Kanten $e \in E$ ausgehend von dem zuletzt entdeckten Knoten u, der immer noch unerforschte Kanten hat. Ähnlich zur Breitensuche wird auch bei der Tiefensuche abgebrochen, sobald der gewünschte Knoten gefunden wurde oder alle Knoten untersucht wurden, um einen Tiefensuchbaum erstellen zu können.

Abb. 3.8 zeigt für einen Graph G die Ergebnisse der Breiten- und Tiefensuche. Angenommen wird, dass die Suche jeweils bei dem rot markierten Knoten startet. Die Ziffern in den Knoten geben an, in welchem Schritt die Knoten besucht werden. Festzuhalten ist an dieser Stelle, dass in diesen einfachen Beispielen die arbiträre Position zweier Knoten in der Adjazenzmatrix herangezogen wird, um über die Sortierung im Falle zweier oder

mehrerer Knoten auf gleicher Ebene beziehungsweise Tiefe zu entscheiden. In Abb. 3.8a wäre daher auch ein Tausch der Knotenpositionen 2, 3 und 4 denkbar, sofern sonstige Kriterien zur Sortierung von Knoten auf gleicher Ebene herangezogen werden würden.



(a) Beispiel einer Breitensuche

(b) Beispiel einer Tiefensuche

Abbildung 3.8: Breiten- und Tiefensuche

Außerdem können auf gerichteten Graphen die *transitive Hülle* sowie die *transitive Reduktion* ermittelt werden. Deren Grundidee liegt in der transitiven Relation. Diese Relation besagt in Kurzform, dass wenn die Relation R zwischen zwei Elementen a und b sowie zwischen b und c gilt, diese auch zwischen a und c existiert:

$$\forall a, b, c \in M : aRb \wedge bRc \Rightarrow aRc \tag{3.7}$$

Bei der Konstruktion einer transitiven Hülle wird für jedes Knotenpaar n_i, n_j analysiert, ob es einen Pfad zwischen diesen Knoten im Graph gibt. Ist dies der Fall, wird eine gerichtete Kante zwischen den Knoten ergänzt. Die *Transitive Reduktion* bezeichnet das umgekehrte Vorgehen. Hierbei werden alle Kanten zwischen zwei Knoten n_i, n_j entfernt, sofern diese anschließend immer noch über einen Pfad miteinander verbunden sind.

Abb. 3.9 illustriert ein Beispiel für den Graph G. Für die transitive Hülle wurde eine neue Kante zwischen Knoten 1 und 5 ergänzt, wohingegen für die transitive Reduktion die Kante zwischen 1 und 5 entfernt wird.

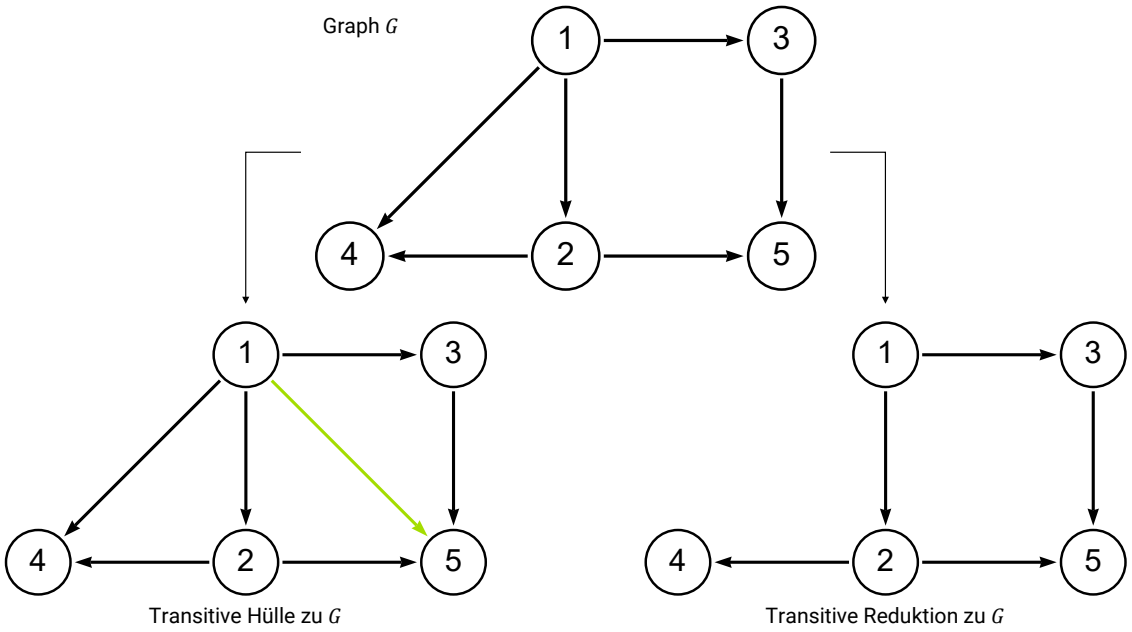


Abbildung 3.9: Ein Graph G mit seiner transitiven Hülle und transitiven Reduktion

3.1.4 Eigenschaften gerichteter Graphen ohne Zyklen

Eine für spätere Ausführungen besonders relevante Gruppe von Graphen stellen *gerichtete Graphen ohne Zyklen* dar. Diese werden unter anderem für viele Probleme der Informatik eingesetzt und in der Literatur oft als Directed Acyclic Graph (DAG) bezeichnet. Sie verfügen über diverse spezifische Charakteristiken, die für die Repräsentation von objektorientierten Datenmodellen als Graph genutzt werden. Als wichtigstes Merkmal eines DAG sei festgehalten, dass ein solcher Graph keine Zyklen enthält. Dies bedeutet, dass keine Möglichkeit existiert, von einem Knoten aus eine endlose Traversierung entlang der gerichteten Kanten zu durchlaufen.

Darüber hinaus kann in diesen Graphen eine *topologische Sortierung* der Knoten vorgenommen werden (Cormen et al., 2009, S. 612–615). Dabei werden die Knoten so angeordnet, dass für jede Kante $e = (u, v)$ der Startknoten u vor dem Zielknoten v auftritt. Anschaulich handelt es sich daher um eine lineare Anordnung aller Knoten des Graphs, sodass alle Kanten in die gleiche Richtung zeigen.

Cormen et al. (2009) geben als einfachen Algorithmus zur topologischen Sortierung das folgende Vorgehen an:

1. Finde einen Knoten ohne eingehende Kanten in G. Wenn mehrere Knoten ohne eingehende Kanten existieren, kann ein beliebiger Knoten gewählt werden.
2. Füge den Knoten in die Liste ein und entferne den Knoten sowie alle zugehörigen (ausgehenden) Kanten.
3. Gebe die verkettete Liste der Knoten zurück.

Diese drei Schritte müssen solange wiederholt werden, bis alle Knoten der Liste hinzugefügt sind und die topologische Sortierung ermittelt ist. Abb. 3.10 zeigt ein Beispiel sowie die notwendigen Zwischenschritte. Der Algorithmus beginnt mit dem Entfernen des Knotens A, der als einziger keine eingehenden Kanten besitzt. Dieser wird der Liste als erstes hinzugefügt. Nach dem Löschen von Knoten A wird Knoten B als nächster Knoten identifiziert, der über keine eingehenden Kanten mehr verfügt. Im gleichen Sinne wird der Graph nach und nach verkleinert bis am Ende kein Knoten übrig bleibt. Sollte sich eine Situation ergeben, bei der kein Knoten mehr gelöscht werden kann, da alle Knoten über eingehende Kanten verfügen, ist der Graph nicht zyklenfrei und damit kein DAG.

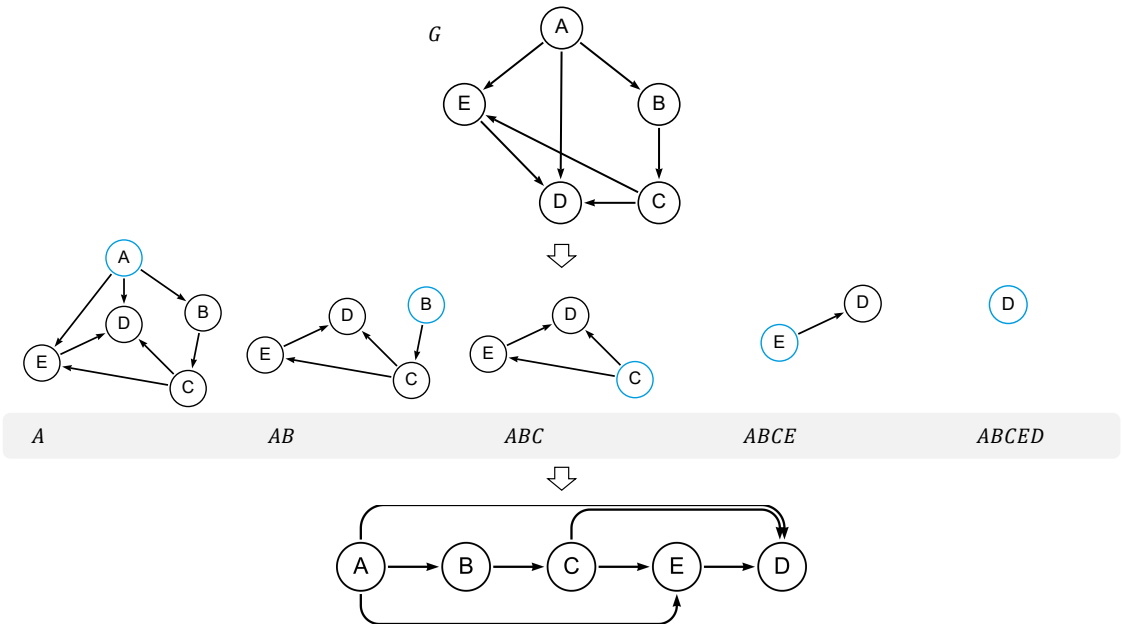


Abbildung 3.10: Beispiel für die topologische Sortierung eines Directed Acyclic Graphs

Die resultierende lineare Anordnung hilft bei der effizienten Organisation und Ausführung von Aufgaben oder Prozessen. Insbesondere in Situationen, in denen die zeitliche oder hierarchische Anordnung von Elementen, die voneinander abhängen, für einen Sachverhalt wichtig sind, kann diese Eigenschaft in der Betrachtung behilflich sein. Die Eigenschaften der DAG werden später für die Betrachtung von Instanzdaten als Graph nochmals aufgegriffen und deren Vorteile geschickt eingesetzt.

3.2 Semantische Graphen zur Beschreibung vernetzter Objektinformationen

Um einen Sachverhalt mit Prinzipien der Graphtheorie beschreiben zu können, müssen passende Annahmen getroffen werden, die wiederum von der zu lösenden praktischen Aufgabenstellung abhängen. Aufgrund dieser Flexibilität finden sich in der Wissenschaft zahlreiche Beispiele, bei denen Graphen zur Beschreibung vielfältigster Probleme herangezogen werden. So beschreibt Foulds (1992), wie die Graphtheorie unter anderem in den Sozial- und Wirtschaftswissenschaften, aber auch in verschiedenen Ingenieurdisziplinen

zum Einsatz kommen. Im Kontext dieser Arbeit sind aber insbesondere Anwendungen von Interesse, die im Bereich der gebauten Umwelt eingesetzt werden. Kolbeck et al. (2022) und Voss et al. (2023) haben eingrenzend Beispiele in Ingenieuranwendungen zusammengefasst. Diese reichen von Überlegungen zu Verkehrs- und Personenstromsimulationen (S. Fischer et al., 2023; Kielar et al., 2018; Kneidl et al., 2012) über den Abgleich von Entwurfs- und Ausführungsinformationen (Kwon et al., 2020) bis hin zu wissensbasierten Entwurfssystemen (Johansson et al., 2018). Flurl et al. (2014) nutzen ein Locking-Konzept in der Kollaborationsplattform, welches dafür sorgt, dass bestimmte Teile eines prozeduralen Graphs gesperrt werden, sobald ein Nutzer ein bestimmtes Objekt bearbeitet.

Darüber hinaus werden Konzepte der Graphtheorie zur Modellierung von Objekten und deren Beziehungen zueinander eingesetzt. Knoten werden dabei gezielt mit semantischen Informationen angereichert, sodass sie für die gewählte Problemdomäne passende Aussagen beinhalten. Zusätzlich können auch Kanten in gewissem Umfang semantische Informationen beinhalten und so verschiedene Beziehungen zwischen Knoten ausdrücken. Graphen dienen dabei vor allem zur Beschreibung bestimmter Zustände einer Domäne und bieten eine hervorragende Grundlage, um Zusammenhänge über mehrere Relationen hinweg zu untersuchen.

Die Überlegungen zur Abbildung eines Domänenzustandes weisen umfangreiche Parallelen zu den Prinzipien der objektorientierten Modellierung auf. Generell sind beide Arten für die Beschreibung vernetzter Informationen geeignet. Die objektorientierte Modellierung umfasst im klassischen Sinne allerdings auch noch Fragestellungen zu Funktionen, Entwurfsmustern und der Anpassung des Modells aufgrund von Berechnungen oder sich verändernden Randbedingungen (beispielsweise verursacht durch Nutzereingaben oder sonstigen angeschlossenen Informationsquellen). Daher ist festzuhalten, dass unter dem Sammelbegriff der Objektorientierten Modellierung (bzw. Programmierung) nicht eine deskriptive Beschreibung des Zustands von verschiedenen Objekten durch ihre Attribute sowie ihrer Wechselwirkungen, sondern auch deren Funktionen und möglicher Mutationen hinzuzuzählen sind. Letzteres ist in den Prinzipien zur Abbildung semantischer Informationen in Graphen nur in sehr begrenztem Umfang enthalten, da sich diese Repräsentationen vor allem auf die Beschreibung durch Attribute und existierender Beziehungen zwischen Objekten konzentrieren. Damit können aber beispielsweise topologische Untersuchungen oder die Suche nach bestimmten Objektmustern deutlich performanter umgesetzt werden als dies auf objektorientierten Strukturen möglich ist.

Hierfür sind derzeit zwei wesentliche Hauptgruppen zu betrachten, die die Abbildung semantischer Information in einer Graphstruktur computerinterpretierbar ermöglichen. Diese seien in den folgenden Abschnitten kurz vorgestellt. Auf der einen Seite existieren dabei Ansätze, die sich auf die Prinzipien von Linked Data und auf das Resource Description Framework (RDF) beziehen. Auf der anderen Seite bietet das Konzept der *beschrifteten Eigenschaftsgraphen* (im Englischen auch als Labeled Property Graph (LPG) gekannt) eine Darstellung, die sehr nahe an den Prinzipien der objektorientierten Modellierung angesiedelt ist.

3.2.1 Beschreibungen für das Semantic Web

Resource Description Framework (RDF)-Graphen stellen Objektnetzwerke in Form von Tripeln dar und werden im Kontext des Semantic Webs eingesetzt. Die Vision vernetzter Informationen und deren Zugänglichkeit über das Internet wurde erstmalig von Tim Berners-Lee beschrieben (Berners-Lee et al., 2001). Seine Idee Anfang der 2000er Jahre beruhte auf der Vision, verteilt gespeicherte Informationen durch bedeutungsreiche Verlinkungen zu vernetzen und so eine verbesserte Entscheidungsgrundlage für vielfältige Probleme zu erreichen. Heute werden die zugehörigen Standards und Empfehlungen insbesondere durch das W3C-Konsortium verwaltet und fortlaufend verbessert.

In einem RDF-Graph besteht jedes Tripel aus einem Subjekt, einem Prädikat und einem Objekt (Sakr et al., 2018). Subjekt und Objekt sind dabei Knoten, das Prädikat wird als Kante im Graph dargestellt. Die Knoten sind daher entweder *Ressourcen*, die ein reales oder abstraktes Objekt repräsentieren, oder *Literale*, die einen Wert und einen Datentyp besitzen. Die Kanten etablieren Beziehungen zwischen zwei Ressourcen oder verknüpfen eine Ressource mit einem *Literal*. Um Knoten eindeutig identifizieren zu können, werden sie mit einem eindeutigen Identifikationsmerkmal ausgestattet (W3C, 2014). Dieses Merkmal wird als Uniform Resource Identifier (URI) bezeichnet und ist in RFC 3986 spezifiziert (Berners-Lee et al., 2005). Das eindeutige Merkmal repräsentiert Informationen, die erforderlich sind, um ein Objekt von allen anderen Dingen innerhalb desselben Identifikationsbereichs zu unterscheiden. Ein URI kann weitergehend in einem sogenannten *Locator* (abgekürzt mit URL) auftreten. Ein *Locator* enthält neben dem Eindeutigkeitsmerkmal ergänzende Informationen, wo die Ressource in einem Netzwerk abgespeichert und über welchen Zugriffsmechanismus sie primär erreichbar ist. Ergänzend haben Dürst und Suignard (2005) in RFC 3987 sogenannte Internationalized Resource Identifier (IRI) beschrieben, die die in URI zugelassene ASCII-Zeichenmenge auf die UTF-Syntaxelemente erweitern. RFC 3987 definiert aus Kompatibilitätsgründen aber ein bidirektionales Mapping, um URI-Merkmale in entsprechende IRI-Darstellungen und umgekehrt zu wandeln.

RDF-Graphen können mit sogenannten RDF-Dokumenten serialisiert ausgetauscht werden. Etablierte Repräsentationen sind dabei Turtle, RDFa, JSON-LD, N3 oder TriG (W3C, 2014). Um Objekte im RDF-Graph weitergehend zu spezifizieren, werden Ontologien genutzt. Ontologien selbst sind ebenfalls RDF-Graphen, repräsentieren aber in der Regel keine Objekte, sondern stellen semantische Informationen bereit, die zur weitergehenden Beschreibung der Objekte genutzt werden.

Abb. 3.11 illustriert einen RDF-Graph, der die Informationen über eine Gebäudetopologie beinhaltet. Die darin dargestellte Information kann beispielsweise in der in Algorithmus 3.1 gegebenen Form als ttl-Datei serialisiert werden.

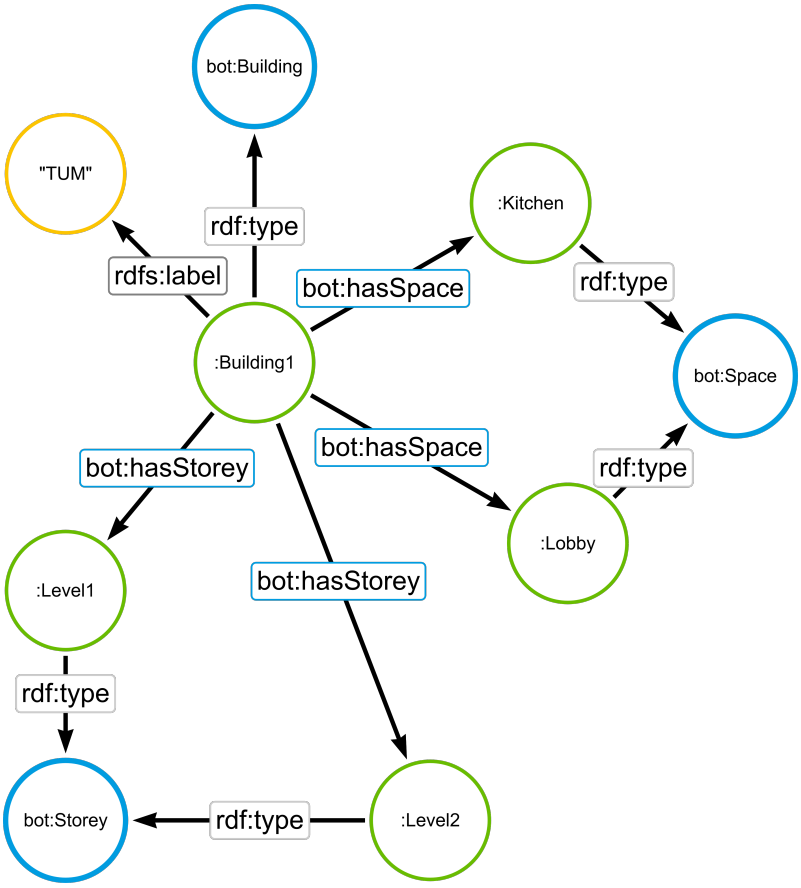


Abbildung 3.11: Beispiel eines RDF-Graphs zur Beschreibung einer Gebäudetopologie mithilfe der Building Topology Ontology (BOT)

Algorithmus 3.1: Beschreibung des in Abb. 3.11 gezeigten Graphs in Turtle-Notation

```
1  @prefix owl:  <http://www.w3.org/2002/07/owl#> .
2  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
4  @prefix bot:  <https://w3id.org/bot#> .
5
6  @prefix : <https://sebesser.de/Sample#> .
7
8  :Level1     rdf:type        bot:Storey.
9  :Level2     rdf:type        bot:Storey.
10
11 :Kitchen    rdf:type        bot:Space.
12 :Lobby      rdf:type        bot:Space.
13
14 :Building1  rdf:type        bot:Building;
15             bot:hasStorey   :Level1,
16                             :Level2;
17             rdfs:label      "TUM";
18             bot:hasSpace    :Kitchen,
19                             :Lobby.
```

Derzeit wird unter der Leitung des W3C-Konsortiums die *RDF-Star* erarbeitet [1]. Ziel dieser Initiative ist die Erweiterung bisheriger RDF-Techniken, um zusätzliche Attribute auch an die verknüpfende Prädikate hinzufügen zu können. Damit werden Beschreibungen möglich, die die Verknüpfung zweier Entitäten betreffen. Um diesem neuen Sprachkonzept gerecht zu werden, wird derzeit zudem die notwendige Erweiterung der Anfragesprache SPARQL erarbeitet.

3.2.2 Beschriftete Eigenschaftsgraphen

Zur Beschreibung strukturierter Instanzdaten, die den Paradigmen der objektorientierten Modellierung folgen, sind beschriftete Eigenschaftsgraphen von besonderem Interesse. Die zugehörigen Konzepte sind unter dem Begriff des Labeled Property Graph (LPG) in der Literatur zu finden und werden im weiteren Verlauf als LPG-Graph bezeichnet (Robinson et al., 2015). Diese Art von Graphen bedient sich den gleichen Grundsätzen wie die RDF-Graphen, bieten aber bereits jetzt zwei weitere wesentliche Modellierungsmöglichkeiten:

1. Knoten und Kanten können mit *Dekoratoren* (im Englischen als *Labels* bezeichnet) versehen werden.
2. Knoten und Kanten können Attributsätze in Form von *Name-Wert-Paaren* tragen, sodass diese nicht als separate Knoten modelliert werden müssen.

LPG-Graphen bieten eine aussichtsreiche Möglichkeit, Informationen über Objekte in wesentlich kompakterer Form darzustellen als dies RDF-Graphen können. Sie erweitern damit das zuvor eingeführte Konzept der Knoten- und Kantenbenennung, welches primär lediglich für eine eindeutige Adressierung ausgewählter Bestandteile eines Graphs zu Erläuterungs- und Illustrationswecken dient. Dekoratoren und Attributsätze bieten darüber hinaus die Möglichkeit, Elemente der Knoten- und Kantenmenge weitergehend zu spezifizieren und deren individuelle Charakteristika direkt an diesen Elementen abzubilden. Ähnliche Funktionalitäten werden derzeit in der Erweiterung RDF-Star für RDF-Graphen diskutiert (Abuoda et al., 2022).

Etablierte Graphdatenbanken

Zur Interaktion mit LPG-Graphen stehen verschiedene Plattformen zur Verfügung, die unterschiedliche Sprachen zur Spezifikation von Mustern sowie der Manipulation der in der Datenbank gespeicherten Graphen verwenden. Fernandes und Bernardino (2018) stellen hierfür verschiedene Produkte gegenüber. Die Anfragesprache *Cypher*, die derzeit vorrangig mit dem Datenbanksystem Neo4j eingesetzt wird, entwickelt sich allerdings derzeit zu einem immer verbreiteteren Quasi-Standard und wird seit kurzem als Open-Cypher entwickelt. Ziel ist laut Francis et al. (2018), diese Anfragesprache zukünftig als ISO-Standard unter der Bezeichnung Graph Query Language (GQL) bereitzustellen [2]. Für

[1] https://www.w3.org/2022/08/rdf-star-wg-charter/ (letzter Zugriff am 11.01.2024)
[2] Weitere Informationen unter https://opencypher.org/ (letzter Zugriff am 08.09.2023)

die weiteren Erläuterungen wird daher nun diese Anfragesprache in ihren Grundzügen weitergehend erläutert.

Wenngleich RDF und LPG in den wesentlichen graphtheoretischen Grundlagen vergleichbare Ansätze verfolgen und strukturierte Informationen flexibel darstellen können, werden die Ansätze zu LPG-Graphen von in manchen Veröffentlichungen kritisch diskutiert (Donkers et al., 2020; Gelling et al., 2023). Neben Performance-Aspekten, die die Ermittlung von Passungen komplexer Muster im Graph betreffen, gibt es insbesondere gegen den schema-freien Ansatz von LPG-Graphen häufig Kritik [3]. So können in zahlreichen LPG-Systemen verschiedener Hersteller Knoten und Kanten mit beliebigem Label angelegt werden. Dies fügt einem System zusätzliche Flexibilität hinzu, bedeutet aber gleichzeitig Abstriche in der konsistenten Datenrepräsentation und der Interoperabilität mit anderen Systemen. Etablierte Systeme wie Neo4j können die Vergabe vorab festgelegter Knotenlabel oder auch verpflichtende Attribute für Knoten mit bestimmtem Label zwar mittlerweile erzwingen, erfordern aber eine tiefgreifende Auseinandersetzung mit den zur Verfügung stehenden Zwangsbedingungen und deren Umsetzung in der Graphdatenbank. Die Verwendung von RDF, passender Ontologien sowie der standardisierten Anfragesprache SPARQL stellen hier eine etabliertere Form zur Beschreibung semantischer Informationen in einem Graph dar, haben aber hingegen weniger Flexibilität in der Erweiterung der genutzten Objektdefinitionen.

Grundlagen der Anfragesprache Cypher

Im Folgenden werden ausgewählte Konzepte der *Cypher*-Anfragesprache vorgestellt, die wie beschrieben zur Interaktion mit LPG-Graphen eingesetzt wird. Sie weist in manchen Prinzipien Ähnlichkeiten zur Structured Query Language (SQL) auf, die seit Jahrzehnten den Standard zur Interaktion mit relationalen Datenbanken bildet.

Der *CREATE*-Befehl dient dazu, definierte Muster in die Graphdatenbank hinzuzufügen. Dabei wird nicht berücksichtigt, ob das spezifizierte Muster bereits innerhalb der Graphdatenbank vorhanden ist oder nicht. Die Spezifikation von Knoten wird durch runde Klammern realisiert. Kanten werden durch eckige Klammern angegeben. Attributsätze werden sowohl für Knoten als auch für Kanten mit geschweiften Klammern ausgedrückt. Die Typisierung von Knoten und Kanten wird durch einen Doppelpunkt eingeleitet. Sofern Knoten und Kanten innerhalb des Musters mehrfach adressiert werden sollen, können zusätzlich Variablen verwendet werden.

Eine beispielhafte *CREATE*-Operation ist in Algorithmus 3.2 gegeben.

Algorithmus 3.2: Erstellung eines Graphs mit Knoten und Beschriftungen sowie Attributsätzen

```
CREATE (n1:A)
CREATE (n2:B:C {Attributname: "AttributWert", NummerischesAttribut: 2.4,
    IstWahr: False, IstFalsch: True})
```

[3] https://opencredo.com/blogs/making-sense-of-data-with-rdf-vs-lpg/ (letzter Zugriff am 11.01.2024)

Erstellt werden mit der in Algorithmus 3.2 gezeigten Operation insgesamt zwei Knoten, die mit Beschriftungen ausgestattet werden. Der Knoten mit Variablenbezeichnung $n2$ erhält dabei die beiden Beschriftungen B und C, wohingegen Knoten $n1$ mit A beschriftet wird. Darüber hinaus wird für Knoten $n2$ ein Attributsatz mit drei Attributen unterschiedlichen Datentyps spezifiziert.

Mit dem *MATCH*-Befehl werden alle Passungen eines spezifizierten Musters p in der Graphdatenbank gesucht. Das Muster kann dabei sowohl topologische Aspekte behandeln als auch Knotenattribute sowie Labels an Knoten und Kanten berücksichtigen. Darüber hinaus kann im Muster spezifiziert werden, ob bei Kanten die Kantenrichtung berücksichtigt werden soll. Die ermittelten Passungen können anschließend in der Abfrage weitergehend prozessiert werden oder als Rückgabewert dienen.

Der Befehl *MERGE* wird einerseits zur Generierung von Beziehungen verwendet und kombiniert andererseits die zuvor eingeführten Befehle *CREATE* und *MATCH*. Zuerst wird in der Graphdatenbank nach passenden Teilgraphen gesucht, die dem definierten Muster entsprechen. Sollten diese aufgefunden werden, so greift der *ON MATCH* Fall. Sollte für das spezifizierte Muster hingegen keine vollständige Passung vorliegen, so wird in Folge das entsprechende Muster zur Graphdatenbank hinzugefügt.

Wichtig ist an dieser Stelle, die Tatsache festzuhalten, dass der *MERGE* Befehl eine *alles-oder-nichts* Passung durchführt. Sollten lediglich Teile des spezifizierten Musters in der Graphdatenbank existieren (formal also eine nicht induzierte Passung zum gewünschten Muster), so wird das gesamte Muster neu generiert. Dieser Aspekt ist insbesondere dann von praktischer Relevanz, wenn zwischen zwei existierenden Knoten eine neue Kante hinzugefügt werden soll. Gegeben sei hierfür beispielhaft der Graph G, der in einer Neo4j-Datenbank vorliegt und in Abb. 3.12 graphisch dargestellt ist. Er besteht zu Beginn aus den zwei Knoten a und b, die jeweils mit dem korrespondierenden Großbuchstaben A beziehungsweise B beschriftet sind.

Algorithmus 3.3: Erstellung des Ausgangsgraphs mit Knoten a und b und leerer Kantenmenge

```
CREATE (a:A), (b:B)
```

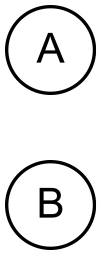


Abbildung 3.12: Graph G mit zwei Knoten und leerer Kantenmenge

Nun soll eine Kante $[ab]$ zwischen diesen Knoten hinzugefügt werden. Naheliegend wäre die Ausführung des in Algorithmus 3.4 gezeigten Cypher-Befehls. Wie beschrieben überprüft der *MERGE*-Befehl zuerst, ob es eine vollständige Passung des spezifizierten Musters gibt. Im konkreten Fall werden also zwei Knoten mit den Beschriftungen A und B gesucht, die zusätzlich mit einer Kante des Typs REL verbunden sind. Da das spezifizierte Muster keine vollständige Passung in der Graphdatenbank finden wird, wird das Muster neu erstellt. Die Systemantwort beinhaltet daher die Neu-Generierung zweier Knoten,

beschriftet mit A und B sowie einer verbindenden Relation des Typs REL. Dies ist in Abb. 3.13 dargestellt.

Algorithmus 3.4: Cypher-Statement zum Einfügen zweier Knoten und einer Beziehung

```
MERGE (a:A)-[:REL]-(b:B)
```

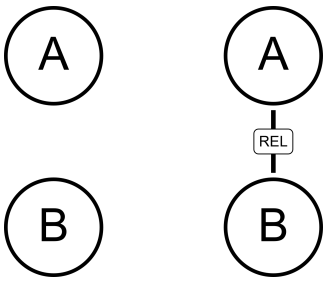


Abbildung 3.13: Ergebnis nach dem Ausführen des reinen MERGE-Befehls

Soll hingegen sichergestellt werden, dass lediglich die neue Kante zwischen den beiden Knoten A und B hinzugefügt wird, so ist eine Kombination aus *MATCH* und *MERGE* notwendig, die in Algorithmus 3.5 dargestellt wird. In der ersten Zeile wird zuerst die notwendige Passung des Knotenmusters gesucht. Erst in der zweiten Zeile erfolgt anschließend die Spezifikation der einzufügenden Kante des Typs REL. Das erzielte Ergebnis zeigt Abb. 3.14.

Algorithmus 3.5: Cypher-Statement zum Einfügen einer Beziehung zwischen zwei existierenden Knoten

```
MATCH (a:A), (b:B)
MERGE (a)-[:REL]-(b)
```

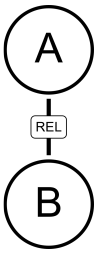


Abbildung 3.14: Ergebnis nach dem Ausführen der Kombination aus MATCH und MERGE

Die Konzepte *MERGE* und *MATCH* zeigen gewisse Ähnlichkeiten zur Projektion in SQL-Statements auf. Analog existiert mit dem *WHERE*-Befehl ebenfalls die Möglichkeit zur Selektion von gefundenen Musterpassungen. Dabei können zum Beispiel weitergehende Anforderungen an Attributwerte oder Wertebereiche definiert werden.

Als weiteres wesentliches Sprachelement kann mit dem *RETURN*-Befehl spezifiziert werden, welche Informationen über den gestellten Datenbankbefehl zurückgegeben werden sollen. Hierbei reichen die Möglichkeiten von der Spezifikation einzelner Knotenattribute bis zur Rückgabe umfangreicher Teilgraphen aus der Datenbank, die dem spezifizierten Muster entsprechen. Durch Zuhilfenahme des Schlüsselwortes *DISTINCT* kann zudem sichergestellt werden, dass die Rückgabewerte vor der Auslieferung an das anfragende System noch normalisiert werden. Mit *LIMIT* ist es möglich, die Anzahl gewünschter Rückgaben zu begrenzen. Hilfreich ist dies beispielsweise, wenn ein gesuchtes Muster mehrfach im Graph auftaucht, aber lediglich die n Passungen zurückgegeben werden sollen. Mit dem *ORDER*-Schlüsselwort kann die Rückgabe zusätzlich geordnet ange-

fordert werden. Sortiert werden kann beispielsweise nach einem bestimmten Knoten- oder Kantenattribut oder aber auch nach der Anzahl von Knoten und Kanten, die in der gewünschten Musterpassung enthalten sind.

Mit der *Cypher*-Abfragesprache können darüber hinaus weitere Konzepte umgesetzt werden, die teilweise in den folgenden Abschnitten verwendet werden. Für deren Dokumentation sei auf die *Cypher*-RefCard verwiesen [4].

3.2.3 Besonderheiten graphtheoretischer Grundlagen in beschrifteten Eigenschaftsgraphen

Wie erläutert ist das Auffinden eines spezifizierten Musters p innerhalb eines Graphs G ein fundamentales Konzept, das unter anderem für die Graphtransformation eine wichtige Rolle spielt. Aufgrund der Tatsache, dass in LPG-Repräsentationen nicht nur topologische Informationen enthalten sind, sondern sowohl Knoten als auch Kanten über umfangreiche Attributsätze verfügen können, müssen weitergehende Überlegungen zur Mustersuche auf dieser Art von Graphen getätigt werden.

Gallagher (2006) unterscheidet verschiedene Möglichkeiten, ein Muster p festzulegen, für das in einem LPG-Graph G Passungen gefunden werden sollen. Für die vorliegende Arbeit sind insbesondere die Unterscheidung in die *topologische Passung* sowie die *semantische Passung* von Relevanz. Allgemein wird unter der topologischen Passung verstanden, dass sich das Muster, nach welchem in einem Graph gesucht werden soll, insbesondere durch die vorliegenden Nachbarschaftsbeziehungen zwischen Knoten und Kanten auszeichnet. Eine semantische Passung bezieht sich hingegen auf die Anwesenheit spezifischer Attribute oder Beschriftungen, die an Knoten oder Kanten vorhanden sein müssen. In Abb. 3.15 wird hierfür ein einfaches Beispiel veranschaulicht.

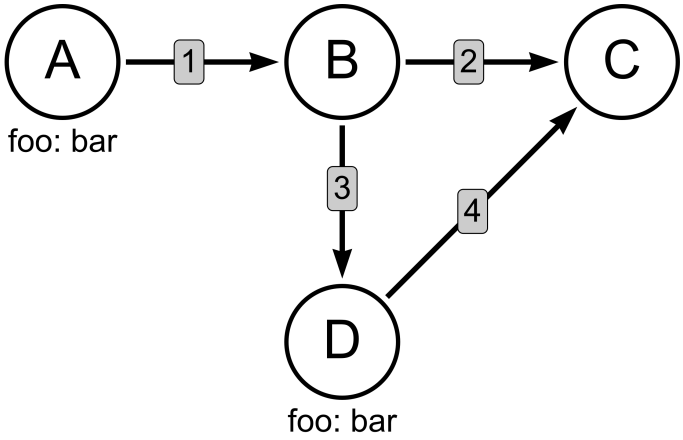


Abbildung 3.15: Beispielgraph G zur Demonstration von topologischer und semantischer Passung eines Musters p

[4] https://neo4j.com/docs/cypher-cheat-sheet/5/neo4j-community/ (letzter Zugriff am 30.11.2023)

Der gegebene Graph G besteht aus den vier Knoten A, B, C und D sowie den gerichteten Kanten 1, 2, 3 und 4. Zusätzlich tragen die Knoten A und D das Attribut foo mit dem Wert $"bar"$.

Betrachtet wird nun zuerst folgende Muster $p1$, das in Algorithmus 3.6 spezifiziert wird.

Algorithmus 3.6: Topologische Mustersuche des Musters $p1$ in Graph G

```
MATCH  p1 = (x)-->(y)-->(z)
RETURN p1
```

Definiert wurde in Algorithmus 3.6 vor allem die Bedingung, dass drei Knoten einen Pfad bilden sollen. Dies erfüllen insgesamt drei Teilgraphen $G' \subseteq G$.

$$(A) \rightarrow (B) \rightarrow (C)$$
$$(A) \rightarrow (B) \rightarrow (D)$$
$$(B) \rightarrow (D) \rightarrow (C)$$

Alternativ kann aber auch nach Passungen des Musters $p2$ gesucht werden, das in Algorithmus 3.7 definiert ist.

Algorithmus 3.7: Semantische Mustersuche des Musters $p2$ in Graph G

```
MATCH  p2 = (x{foo: "bar"})
RETURN p2
```

In diesem Falle werden alle Knoten gesucht, die über das Attribut foo verfügen. Zusätzlich muss das Attribut mit dem Wert $"bar"$ belegt sein. Als Resultat sind zwei Passungen zu erwarten:

$$(A)$$
$$(D)$$

Selbstverständlich sind auch Kombinationen aus topologischen und semantischen Mustern möglich. Hierfür werden die vorangegangenen Beispiele kombiniert und das Muster $p3$ in Algorithmus 3.8 formuliert.

Algorithmus 3.8: Kombinierte topologische und semantische Mustersuche des Musters $p3$ in Graph G

```
MATCH  p3 = (x{foo: "bar"})-->(y)-->(z)
RETURN p3
```

Neben den Bedingungen zu den adjazenten Knoten, verbunden durch je eine gerichtete Kante wird nun eingefordert, dass der Knoten x zusätzlich über das Attribut foo mit dem Wert bar verfügt. Als Resultat sind dabei zwei Passungen zu erwarten:

$$(A) \rightarrow (B) \rightarrow (C)$$
$$(A) \rightarrow (B) \rightarrow (D)$$

Erwähnenswert ist hierbei, dass der Teilgraph aus den Knoten $A \rightarrow B \rightarrow D$ nach wie vor eine Passung zum in Algorithmus 3.8 spezifizierten Muster darstellt. Der Knoten D verfügt ebenfalls über das Attribut $foo : "bar"$. Da die Musterspezifikationen aber immer nur Mindestanforderungen stellen, können Knoten für eine positive Passung auch weitere Attribute tragen. Möglich wäre, auch Negativ-Bedingungen zu definieren und einzelne Passungen damit auszuschließen. Weiter zeigt das Ergebnis zum in Algorithmus 3.8 definierten Muster eine Passung weniger als jenes, welches zum in Algorithmus 3.6 spezifizierten Muster angegeben ist. Der Teilgraph $B \rightarrow D \rightarrow C$ erfüllt das Muster nicht, da der Knoten B das geforderte Attribut $foo : "bar"$ nicht aufweist.

3.2.4 Beispiele für die Verwendung von Graphen im Kontext des Bauwesens

Die theoretisch eingeführten Arten von Graphen zur Modellierung vernetzter Informationen werden in der Bauinformatik für unterschiedliche Anwendungen genutzt, die unter anderem in Pauwels et al. (2022) umfangreich beschrieben sind. Beetz et al. (2009) und Pauwels und Terkaj (2016) beleuchten Ansätze, wie das IFC-Datenmodell als Ontologie dargestellt werden kann. Insbesondere der Umgang mit sortierten Listen erfordert dabei einen erheblichen Aufwand, da diese als Binärbaum beschrieben werden müssen. Amann et al. (2021) führen aus, dass für deren Abbildung *First-Rest-Paare* gebildet werden, um die exakte Reihenfolge zu erhalten. Alternative Lösungen für diese Darstellungsart wurden unter anderem von Pauwels et al. (2015) vorgeschlagen. Diese umfassen einerseits die Umgestaltung von Klassen ohne geordnete Aggregate und andererseits Überlegungen, geometrische Konzepte in Well-Known-Text (WKT) zu formulieren. Bei WKT handelt es sich um eine Auszeichnungssprache, die insbesondere für die Beschreibung von geometrischen Formen und deren Austausch zwischen Geoinformationssystemen eingesetzt wird (ISO, 2019b).

Rasmussen et al. (2020) kritisieren an der direkten Abbildung des IFC-Datenmodells als Ontologie wesentliche Limitationen, das bijektive Mapping des gesamten Datenmodells mit sich nur geringe Vorteile birgt. Bemängelt wird insbesondere die entstehende Komplexität, die ifcOWL aufweist. Um eine verlustfreie Übersetzung zwischen den Konzepten des EXPRESS-Schemas und der Ontologie-Repräsentation zu ermöglichen, wurden diverse Kompromisse in der Definition der Ontologie getroffen. So müssen spezielle syntaktische Konstrukte verwendet werden, um beispielsweise objektifizierte Beziehungen oder sortierte Listen als RDF-Graph abzubilden. Weitergehend kritisieren Rasmussen et al. (2020) die Größe der Ontologie und die damit verbundenen Einschränkungen in der Modularität und der Erweiterbarkeit. Für die IFC Version 4 berichten die Autoren von 1331 Klassen sowie 1599 Attributen, die die ifcOWL-Ontologie spezifiziert. Sowohl die eingesetzten Konstrukte als auch die große Zahl an Begriffen widerspricht der Grundidee von Ontologien, einen Sachverhalt mit möglichst wenigen angemessenen Klassen und Attributen zu beschreiben.

Neben den Ansätzen zur Abbildung des IFC-Datenmodells als Ontologie wurde die Anwendung von RDF-Technologien für die Beschreibung gebauter Umwelt von verschiedenen Forschungsgruppen vorangetrieben, die Teilaspekte des IFC-Datenmodells berücksich-

tigen und passende Referenzen ermöglichen. Zu nennen sind dabei unter anderem die BOT-Ontologie, die passende Begriffe zur Beschreibung von Gebäudetopologien bereitstellt (Rasmussen et al., 2020), oder die BPO-Ontologie, die zur Beschreibung von Bauprodukten genutzt werden soll (Wagner et al., 2022).

Heute existieren zahlreiche weitere Ontologien, die spezifische Aspekte der Planung, des Baus und des Betriebs von Gebäuden und Infrastrukturanlagen und zugehöriger Anwendungsfälle berücksichtigen. Göbels (2022) verwendet beispielsweise RDF-Repräsentationen für die Bestandsmodellierung für Infrastrukturanlagen. Ebenso stehen Tripel-basierte Darstellungen verschiedener Informationsressourcen in den Ausführungen von Schlenger et al. (2022) im Mittelpunkt und werden in Kombination mit verschiedenen, bestehenden Ontologien verwendet, um die Bausführung in einer prozessorientierten Weise beschreiben zu können. Donkers et al. (2023) nutzen Ontologie-Repräsentationen für die Aufbereitung von Sensordaten in Gebäuden, wohingegen Guyo et al. (2023) spezifische Definitionen für Evakuierungs- und Brandszenarien in Gebäuden als Ontologie bereitstellen. Darüber hinaus hat Bonduel (2021) durch die Verknüpfung und Erweiterung verschiedener Ontologien ein umfassendes System zur semantischen Beschreibung historischer Bauwerke entwickelt. Teclaw et al. (2023) verwenden verschiedene der genannten Ontologien sowie zusätzliche geometrische Eigenschaften, um Objekte verschiedener Disziplinmodelle in Beziehung zu bringen. Ihre Studie hatte in erster Linie zum Ziel, äquivalente Räume und Ebenen in verschiedenen Disziplinmodellen zu identifizieren. Obwohl ihr Ansatz für deren Fallstudie aussagekräftige Ergebnisse gezeigt hat, sind die gewählten Kriterien zur Ermittlung äquivalenter Modellobjekte schwer zu verallgemeinern.

Ergänzend zu den Bemühungen, Linked-Data Ansätze zu nutzen, gewann auch die Anwendung von LPG-Graphen zur Beschreibung von IFC-Modellen an Relevanz. Ismail et al. (2017) haben eine Methode zur Abbildung von IFC-Modellen auf die Graphdatenbank Neo4j entwickelt. Dabei werden insbesondere die effizienten Filtermöglichkeiten auf Basis der graphbasierten Darstellung der IFC-Instanzen als LPG hervorgehoben. Vergleichbare Überlegungen haben auch Zhu et al. (2023) angestellt. Sie heben dabei insbesondere die im Vergleich zu entsprechenden RDF-Repräsentationen kompakte Struktur ihres IFC-Graphs hervor, behandeln ausgehend davon aber keine weiteren Aspekte, die sich durch die Wandlung der in einem IFC-Modell gegebenen Informationen in eine Graphstruktur ergeben könnten. Tauscher et al. (2016) verwenden ebenfalls Graphrepräsentationen, um generische Abfragen direkt auf den in einem IFC-Modell enthaltenen Objekten durchführen zu können. Die angegebenen Abbildungsmechanismen zwischen textbasierter SPF-Darstellung und resultierendem Graph bedienen sich im wesentlichen den Konzepten, die auch in den anderen Veröffentlichungen aufgeführt werden. Im Grundsatz folgen alle Ansätze den Überlegungen von Hidders (2001), der einen allgemeinen Ansatz zur Abbildung objektorientierter Datenstrukturen auf Graphrepräsentationen beschreibt. Außerdem sind in diesem Zuge die Überlegungen zur Abbildung von Gebäudemodellen auf relationale Datenbanksysteme zu nennen (Nour, 2009). Motiviert wurden diese unter anderem durch die Perspektive, BIM-Modelle mit der etablierten Abfragesprache SQL

bedienen zu können. Diese werden im Rahmen der vorliegenden Arbeit aber nicht weiter untersucht, da sich die Anwendung graphbasierter Systeme als geeigneter zeigt.

In der jüngeren Vergangenheit wurden Graphrepräsentationen auch für verschiedene Verfahren herangezogen, die Methoden der Künstlichen Intelligenz zur Analyse oder Aufbereitung von Informationen der gebauten Umwelt nutzen. Besonders relevant sind dabei unter anderem so genannte *Graph Neural Networks*, die vor allem für Klassifikations- und Vorhersageprobleme eingesetzt werden. Collins et al. (2022) stellen ein Verfahren zur Klassifikation von Modellelementen auf Basis einer graphbasierten Kodierung von Geometrien vor. Austern et al. (2024) behandeln ein ähnliches Klassifikationsproblem, versuchen allerdings durch eine zusätzliche Beschreibung direkt angrenzender Komponenten eines Bauteils die Ergebnisse weiter zu verbessern. Darüber hinaus werden Graphrepräsentationen häufig als Ausgangspunkt für generative Verfahren eingesetzt, um aus einem Ausgangszustand möglichst viele Varianten erzeugen zu können. Diese werden anschließend basierend auf vordefinierten Kritierien bewertet und gegebenenfalls weiter optimiert, falls noch keine ausreichend gute Lösung erzielt wurde. Ansätze, wie Graphrepräsentationen für solche Überlegungen in der Suche nach optimierten Raum- und Gebäudeformen eingesetzt werden können, sind unter anderem in Lin et al. (2024) dokumentiert worden.

Neben Abbildungen von Instanzdaten als Graphstrukturen sind verschiedene Überlegungen dokumentiert, die auf die Prinzipien der Graphersetzung zurückgreifen. So beschreibt Vilgertshofer (2022) beispielsweise parametrische Geometrien als LPG-Graph und modifiziert diese Repräsentation durch geeignete Graphersetzungsregeln. Dafür wird ein speziell für dieses Problem definiertes Graph-Metamodell herangezogen, das über Definitionen für geometrische Formen und Körper im zweidimensionalen und dreidimensionalen Raum verfügt. Im Weiteren werden eine Reihe von dimensionalen und geometrischen Zwangsbedingungen definiert, die verschiedene Geometrien in Relationen setzen können. Durch Anwendung von Graphersetzungsregeln können anschließend verschiedene geometrische Operationen produktunabhängig beschrieben werden, sodass prozedurale Beschreibungen auch über die Grenzen spezifischer Produkte hinweg ausgetauscht werden können. Für die prototypische Umsetzung führt Vilgertshofer verschiedene Produkte auf, von denen er letztendlich das Graphersetzungswerkzeug GrGen eingesetzt hat (Jakumeit et al., 2010; Vilgertshofer & Borrmann, 2017).

3.3 Versionskontrollsysteme für Graphrepräsentationen

Die zuvor beschriebenen Ansätze zur Darstellung der einem Modell zugrundeliegenden Objektstruktur als Graph wurde in den zitierten Werken vor allem durch die gesteigerte Flexibilität motiviert. Weitgehend unbehandelt ist aber die Frage, wie diese Graphrepräsentationen im Falle von Änderungen am zugrundeliegenden Modell erkannt und beschrieben werden können.

Berners-Lee und Connolly (2004) beschrieben ein Grundprinzip, wie Änderungen zwischen zwei RDF-Graphen ermittelt und auf eine veraltete Version wieder angewendet werden konnte. Meinhardt et al. (2015) griffen diese Überlegungen auf und stellen in ihrer Arbeit verschiedene Versionierungssysteme für RDF-basierte Datenrepräsentationen vor. Die Autoren motivieren ihre Bemühungen mit den Herausforderungen, die sich bei dynamisch verändernden RDF Graphen ergeben. Hierzu haben sie ein Versionsmanagementsystem entwickelt, bei dem ein Repository mehrere Paare aus RDF-Datensätze und URI-Merkmalen sowie zugehöriger Versionen verwalten kann. Als atomare Modifikationen werden weiter das Einfügen eines neuen RDF-URI-Paares, das Entfernen eines solchen Paares oder das Ändern eines bestehenden Paares festgelegt. Soll nun eine Anfrage auf einer bestimmten Version erfolgen, wird als zusätzlicher Parameter der gewünschte Zeitpunkt des zu untersuchenden Datensets mit angegeben. Das Versionskontrollsystem kann die Anfrage anschließend auf dem korrespondierenden RDF-URI-Paar ausführen und die gewünschten Informationen passend ausliefern. Limitierend wird erwähnt, dass in der beschriebenen Ausbaustufe des Systems noch keine versionsübergreifenden Abfragen möglich sind. Vergleichbare Betrachtungen haben auch Papavasileiou et al. (2013) angestellt.

Rasmussen erarbeitete Ontologien, mithilfe derer sich die Weiterentwicklung von RDF-basierten Wissensrepräsentationen abbilden lässt (Rasmussen et al., 2018, 2019). Hierfür werden der Graphstruktur mit jeder Aktualisierung ein weiteres Paket an Tripeln hinzugefügt, die den neuen Zustand beschreiben. Ein sehr verwandtes Vorgehen ist beispielsweise auch in einzelnen Objektdatenmodellen vorgesehen. Im IFC-Standard dient dabei die `IfcOwnerHistory`-Entität zur Erfassung des Autors, der die letzte Änderung an einem Objekt vorgenommen hat. Da die Verknüpfung der `IfcOwnerHistory` ausschließlich mit Entitäten des Root Layers möglich ist, können die Veränderungen einzelner Attribute nicht feingranular erfasst werden. Abhängig vom konkreten Anwendungsfall erscheint demnach der Ansatz von Rasmussen mit einer Ontologie als allgemeingültiger. Weitere Hinweise zu Implementierungen, die zwei Versionen von RDF-Graphen vergleichen können, sind auf einer Webseite des W3C zu finden [5].

Nuha (2019) erläutert in seiner Arbeit allgemeine Herausforderungen des Versionsmanagements von Datensätzen, die beispielsweise in Prozessen des maschinellen Lernens genutzt werden. Eine besondere Herausforderung in diesem Bereich sei es nach seinen Ausführungen, erzielte Ergebnisse reproduzierbar mit den verwendeten Trainings- und Validierungsdaten zu machen. Zu diesem Zweck hat Nuha eine Applikation zur Versionsverwaltung des Graphdatenbanksystems neo4j entwickelt. Hierzu wird eine Überwachung an der Graphdatenbank vorgesehen, die die auszuführenden Befehle abfängt. Diese werden anschließend in einem *ChangeLog* gesammelt. Möchte der Nutzer einen Zustand als Version speichern, werden alle erfassten Befehle ausgewertet und gegebenenfalls kompensiert, sofern spätere Befehle zuvor erfolgte Transaktionen obsolet machen. Als Limitation des Ansatzes bleibt die Bindung an die Programmierschnittstelle festzuhalten, die zur Überwachung aller eingehenden Befehle erfolgt. Diese kann in alternativen Produkten

[5] https://www.w3.org/2001/sw/wiki/How_to_diff_RDF (letzter Zugriff am 26.01.2024)

und Systemen eine andere Gestalt aufweisen oder nicht als offene Schnittstelle verfügbar sein. Darüber hinaus kann das Versionskontrollsystem nicht überwachen, ob ein erfasster Befehl eine erfolgreiche Transaktion ausgelöst hat. Demnach können im *ChangeLog* auch fehlerhafte und für die Datenbank uninterpretierbare Informationen enthalten sein.

3.4 Einordnung

Aus den aufgeführten Arbeiten geht hervor, dass in den letzten Jahrzehnten umfangreiche Anstrengungen unternommen wurden, um Objekte und ihre Wechselwirkungen in Formen zu beschreiben, die die Erfassung und den Austausch zwischen den für das Bauwesen konzipierte Softwaresystemen ermöglichen. Neben den in Kapitel 2 vorgestellten Grundprinzipien zu Datenmodellen und deren Anwendung im Bauwesen zeigen sich die Methoden zur Beschreibung semantischer Informationen in Graphen als geeignetes Mittel, Phänomene und Limitationen zu überwinden, die durch die existierenden Serialisierungsstrukturen induziert werden. Gleichzeitig ist zu beobachten, dass die in Abschnitt 2.3.5 erläuterten Datenmodelle zunehmende Unterstützung in vielfältigen kommerziellen und frei zugänglichen Softwareapplikationen erfahren. Es erscheint daher nicht sinnvoll, sämtliche Informationen zukünftig nur noch als Graphdarstellungen zu behandeln und damit die Notwendigkeit neuer Import- und Exportschnittstellen zu erzeugen, die Informationen aus Autorensystemen direkt als Graphrepräsentation bereitstellen. Vielmehr können existierende Schnittstellen weiter genutzt werden, um die notwendigen Repräsentationen aus den Export-Produkten zu generieren. Zugleich ist festzuhalten, dass neben dem IFC-Datenmodell weitere Spezifikationen eingesetzt werden, die insbesondere in spezialisierten Disziplinen weiterhin ihre Daseinsberechtigung besitzen. Daher ist es notwendig, das anvisierte Versionskontrollsystem für Produktdatenmodelle möglichst generisch zu gestalten, um es flexibel für verschiedene Datenmodelle einsetzen zu können. In den nächsten Kapiteln wird daher nun eine objektbasierte Versionskontrollmethode eingeführt, die BIM-Modelle in ihrer zugrundeliegenden Objektstruktur behandelt, diese in passende Graphrepräsentationen wandelt und anschließend die bereits theoretisch eingeführten Grundlagen der Graphtransformation nutzt, um Versionsinkremente zu ermitteln.

Kapitel 4

Entwicklung einer graphbasierten Versionskontrollmethodik für BIM-Modelle

Aus den dargelegten Betrachtungen geht hervor, dass es für BIM-Modelle verschiedene Ansätze für deren Repräsentation in herstellerneutralen Datenmodellen und Serialisierungsmechanismen gibt, die in Projekten teils parallel eingesetzt werden. In Abschnitt 2.8 wurden zudem die Implikationen der erläuterten Serialisierungsstrategien und der damit verbundenen Defizite bei dem Einsatz klassischer textbasierter Versionskontrollsysteme für die Verwaltung von vernetzten Objektinformationen diskutiert. Diese bieten folglich nur unter umfangreichen Annahmen eine sinnvolle Basis, um Modifikationen auf Basis der tatsächlich veränderten Objekte zwischen Projektbeteiligten zu übertragen.

In diesem Kapitel wird nun der im Rahmen dieser Arbeit entwickelte Systemansatz vorgestellt, der die benannten Probleme überwindet und Änderungen an BIM-Modellen auf Basis der zugrundeliegenden vernetzten Objektstrukturen beschreibt. Grundsätzlich sollen folgende Ziele erfüllt werden:

- Entwicklung eines Systems zur objektbasierten Versionskontrolle, das unabhängig von dem zugrundeliegenden Datenmodells ist.
- Erzielen der Unabhängigkeit von der gewählten Serialisierungsform und -reihenfolge.
- Gewährleisten der Anwendbarkeit für möglichst viele objektorientierte Datenmodelle, die im Bereich des Bauwesens derzeit eingesetzt werden.
- Direkte Zugänglichkeit zu den erfassten Modifikationen in Inkrementen.
- Sicherstellung der vollständigen und verlustfreien Übermittlung von Inkrementen sowie deren Anwendung auf veraltete Modellversionen.

Angenommen wird an dieser Stelle, dass die untersuchten Datenmodelle den ingenieurmäßigen Ansprüchen genügen und passende Beschreibungen für den darzustellenden planerischen Sachverhalt bereitstellen. Die anvisierten Funktionalitäten sollen diese Definitionen um eine Methodik zur inkrementellen Versionskontrolle von Objekten auf MOF-M0-Ebene ergänzen. Eine Erweiterung der Datenmodelle um zusätzliche Klassen oder Attribute soll vermieden werden, um die direkte Anwendbarkeit der entwickelten Methode in bestehende Prozesse und etablierte Softwareprodukte zu gewährleisten.

4.1 Randbedingungen und Abwägungen zur Umsetzung einer objektbasierten Versionskontrollmethode

Abb. 4.1 illustriert verschiedene Aspekte, die im Zuge der Entwicklung berücksichtigt wurden. Diese Randbedingungen werden in der Folge erläutert und daraus passende Prinzipien für die Versionsüberwachung abgeleitet.

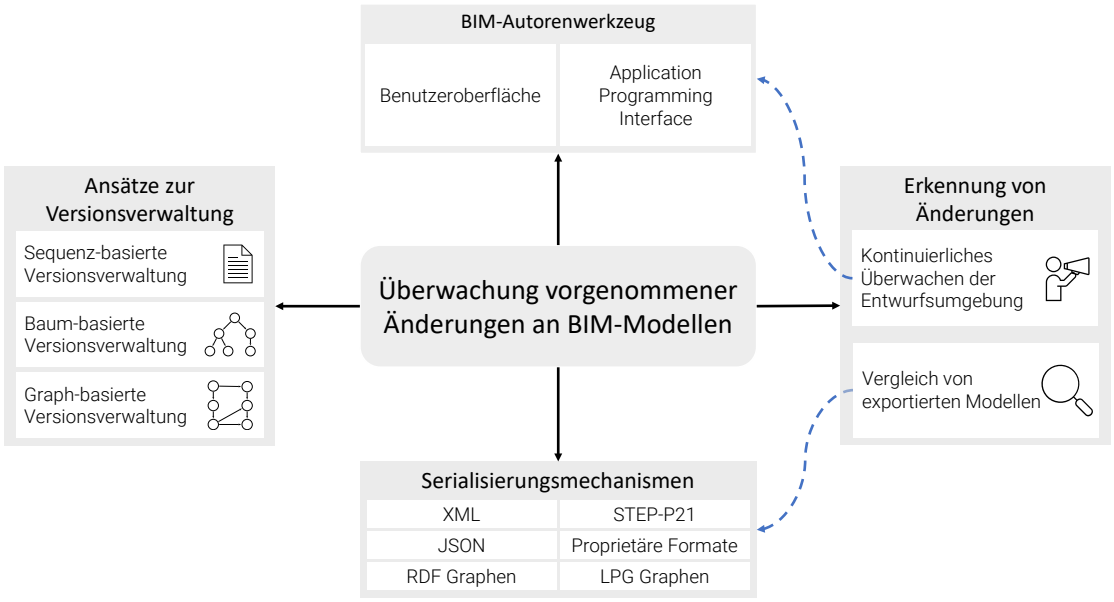


Abbildung 4.1: Aspekte und Überlegungen zur modellbasierten Versionskontrolle

Zu Beginn der Betrachtung steht eine menschliche Ressource, die durch ihr Wissen eine Entwurfsapplikation bedient und damit planerische Überlegungen in ein geeignetes Modellierungssystem eingibt. Die somit abstrahierten Informationen werden in einem weiteren Schritt in die Form passender Disziplinmodelle gefasst, die die Konzepte der BIM-Methodik verfolgen und unterstützen. Im Kern beschreibt ein Disziplinmodell allerdings ein komplexes Geflecht aus verschiedenen Objekten, die in ihrer Gesamtheit das BIM-Modell formen. Formal handelt es sich dabei um MOF-M0 Objekte, die Instanzen eines zugehörigen Datenmodells aus der MOF-M1 Ebene darstellen. Darüber hinaus müssen diverse Aspekte in die Überlegungen und die Auslegung des Versionskontrollsystems einfließen. Weiter müssen Beschränkungen bestehender Systeme betrachtet werden, woraus sich weitere Randbedingungen für den entwickelten Ansatz ableiten.

Die in Abschnitt 2.3.5 vorgestellte Analyse verschiedener Produktdatenmodelle im Kontext des Bauwesens hat verdeutlicht, dass alle untersuchten Datenmodelle für herstellerneutralen Informationsaustausch den wesentlichen Grundprinzipien der objektorientierten Modellierung folgen. Wie bereits in Abschnitt 2.3.4 beschrieben existieren für die Objektstrukturen verschiedene Mechanismen zur Serialisierung mit gängigen Auszeichnungssprachen, die auch bei dem Import und Export von BIM-Modellen zum Einsatz kommen. Folglich wäre der Einsatz inkrementeller Verfahren für textuelle oder hierarchisch

strukturierte Repräsentationen denkbar. Wie in Abschnitt 2.5.3 vorgestellt gibt es hierfür eine Reihe von Ansätzen, die Änderungen zwischen zwei XML- oder JSON-Dokumenten ermitteln und in Form einer Änderungsprozedur repräsentieren können. Diese Verfahren unterscheiden allerdings die verschiedene Arten von Verweisen, die in einem Dokument auftreten können. So wird das Hinzufügen von Kindelementen als strukturelle Änderung behandelt, während attributbasierte Querverweise lediglich als Modifikation einer Objekteigenschaft gewertet werden. Im speziellen Fall von STEP-Part 21 kodierten Dateien existiert diese Unterscheidung nicht. Weiter gibt es hierfür wie in Abschnitt 2.4 erläutert bisher keine Versionskontrollmechanismen, die eine allgemeine und vollständige Überwachung solcher Repräsentationen ohne vorangestellter Annahmen ermöglichen.

Um diesen Einschränkungen adäquater zu begegnen, wird die Anwendung von Prinzipien der Graphtransformation vorgesehen, wie sie in Abschnitt 3.1.2 bereits theoretisch eingeführt wurden. Damit einher geht die Notwendigkeit, eine geeignete Graphrepräsentation für die MOF-M0 Instanzdaten zu finden, um die Informationen des BIM-Modells verlustfrei in einem Graph abbilden zu können.

Weitergehend muss ein strategischer Ansatz gewählt werden, wie man Kenntnis über die auf Modelle angewandten Modifikationen erlangen kann. Eine Möglichkeit zur Ermittlung vorgenommener Änderungen an einem BIM-Modell ist das kontinuierliche Überwachen der eingesetzten Autorenwerkzeuge. Aus den erfassten Nutzereingaben und den damit verbundenen Transaktionen auf den internen Datenstrukturen des Autorenwerkzeuges können passende Transformationsregeln für die Austausch-Modelle abgeleitet werden. Voraussetzung für diesen Ansatz ist, dass die Entwurfsumgebung durchgehend überwacht werden kann und passenden Zugang zu den notwendigen Informationen bereitstellt. Aus dem Blickwinkel der Softwareentwicklung werden solche Mechanismen mit dem so genannten Beobachter-Muster (im englischen *Observer Pattern*) realisiert. Die Kernidee hinter diesem Entwicklungsparadigma ist die Benachrichtigung von anderen Funktionalitäten in einem Programm, wenn ein überwachtes Objekt modifiziert wird. Sobald der Zustand eines überwachten Objekts verändert wird, erfolgt die Aussendung einer Nachricht an alle Beobachter. Diese wiederum können passende Logik zur Verarbeitung der Änderungsinformation vorhalten und im Falle des Änderungsereignisses auslösen (Eilebrecht & Starke, 2019, S. 70–75).

Der Einsatz des Beobachter-Musters ermöglicht das direkte Aufzeichnen vorgenommener Modifikationen. Dabei ergeben sich allerdings signifikante Randbedingungen. Grundsätzlich muss das eingesetzte Entwurfswerkzeug über passende Schnittstellen verfügen, um Nutzereingaben überwachen zu können. Wenngleich die Beschreibung des generellen Verhaltens in Autorenwerkzeugen sehr ähnlich ist, unterscheiden sich die konkreten Methoden und Interfaces je nach Hersteller und Applikation dennoch in Benennung und tatsächlich verfügbarer Funktionalität. Beschränkend kommt hinzu, dass die Zugriffsmöglichkeiten auf das zugrundeliegende applikationseigene Datenmodell über die Entwicklungsschnittstellen meist beschränkt sind oder beispielsweise aus Gründen des Innovationsschutzes gar nicht angeboten werden. Es wäre daher nötig, für jedes verwendete Autorenwerkzeug

eigene Plugins zu entwickeln, um die entsprechende Überwachung der programminternen Speicherstrukturen zu realisieren.

Darüber hinaus werden die Benachrichtigungen über die Änderung von Objektzuständen meist erst dann ausgelöst, wenn die programminterne Speicherstruktur bereits erfolgreich verändert wurde. Insbesondere bei Vorgängen, bei denen Objekte gelöscht werden, stellt dies ein Hindernis für die Ableitung entsprechender Transformationen auf den herstellerneutralen Repräsentationen dar, da die interne Datenstruktur über die gelöschten Objekte nur noch sehr begrenzte Informationen bereitstellt. Dies erschwert die Ableitung passender Modifikationen erheblich, die im Export-Derivat vorzunehmen wären. Die generelle Philosophie, Benachrichtigungen an Beobachter erst nach erfolgreichen Transaktionen auf den internen Speichern auszulösen, ist dabei allerdings nachvollziehbar, um im Falle unvollständiger Transaktionen unnötigen Aufwand bei den Beobachtern eines Objekts zu verhindern.

Eine zusätzliche Schwierigkeit liegt darin, dass die Speicher- und Objektstruktur, die intern von BIM-Autorenwerkzeugen verwendet wird, von den Datenstrukturen abweicht, die für den Datenaustausch verwendet werden. Insbesondere Modellierungswerkzeuge für komplexe geometrische Formen nutzen meistens prozedurale oder parametrische Repräsentationen. Für den Datenaustausch mit anderen Projektparteien werden diese Strukturen in vielen Fällen in Repräsentationen übersetzt, die von der korrespondierenden programminternen Repräsentation mehr oder weniger deutlich abweichen können. Diese Verarbeitung ist unter anderem erforderlich, um bestimmte Vereinfachungen komplexer Strukturen vorzunehmen oder sie in eine Form zu transformieren, die zu dem gewünschten Datenmodell passt, mit dem ein Datenaustauschszenario realisiert werden soll. Umgekehrt sind auch Fälle denkbar, bei denen bei der Ausführung des Übersetzungsprozesses von internen Informationen in offene Datenmodelle zusätzliche Informationen erhoben und dem Export-Resultat hinzugefügt werden müssen. In beiden skizzierten Szenarien gestaltet es sich als komplex, klare Regeln und Funktionen aufzustellen, um eine direkte Verbindung zwischen überwachten Änderungen in den internen Datenstrukturen eines Autorensystems und den gewünschten Darstellungen in den Übertragungsformaten herzustellen. In den meisten Fällen handelt es sich nicht um bijektive Abbildungen, bei denen eine Modifikation an der Programm-internen Datenstruktur genau einer Transformation entspricht, die auf das Exportergebnis angewendet werden kann.

Als Alternative zu einer dauerhaften Überwachung eines Autorensystems bietet es sich an, verschiedene Versionen eines BIM-Modells zuerst mit bestehenden Export-Schnittstellen des BIM-Autorenwerkzeugs zu speichern und ausgehend von diesen anschließend die vorgenommenen Änderungen zwischen zwei Versionen des Modells zu bestimmen. Da bei diesem Ansatz auf bestehende Schnittstellen der Autorensysteme zurückgegriffen wird, bedarf es in diesem Fall keiner individualisierten Implementierung für jedes Softwareprodukt, das an einem Datenaustausch teilnimmt. Damit entfällt auch die Problematik, wie interne Änderungen auf die Austauschrepräsentationen eines Modells erfasst werden.

Bei diesem Ansatz entstehen vorübergehend mehrere Dateirepräsentationen des BIM-Modells auf der sendenden Seite, die bis zur Bestimmung des Iinkrements im System gespeichert werden müssen. Darüber hinaus gestaltet sich die Identifizierung der modifizierten Elemente zwischen zwei Modellversionen als aufwendiger, da dafür kein Vorwissen aus den Nutzereingaben oder dem Autorensystem genutzt werden kann. Positiv ist jedoch zu bewerten, dass der Nutzer den Zeitpunkt für die Auslösung des eigentlichen Versionierungsprozesses aktiv und eigenständig bestimmen kann.

4.2 Gewählter Ansatz und Vorgehen

Durch die Anwendung der graphentheoretischen Grundlagen können sämtliche Aspekte berücksichtigt werden, die sowohl in baumartigen als auch beliebig vernetzten Datenstrukturen auftreten. Daher werden diese Ansätze im Folgenden weiter verfolgt und mögliche Vereinfachungen, die sich aufgrund ihrer hierarchischen Form ergeben würden, nicht weiter beleuchtet. Da hierarchisch angeordnete Objektstrukturen in vielen Belangen Vereinfachungen von Konzepten der Graphentheorie darstellen, wird durch die Entscheidung, vorrangig Graphen zu betrachten, die allgemeine Anwendbarkeit gewährleistet. Dabei sind die im Folgenden dargelegten Überlegungen einerseits von den identifizierten Limitationen aus Abschnitt 2.5.3 als auch von den Überlegungen von Chawathe et al. (1996) zur Abstraktion von Änderungen in hierarchischen Repräsentationen vorgegangen inspiriert. Deren Erläuterungen beruhen aber überwiegend auf der Untersuchung hierarchisch angeordneter Daten, sodass für die derzeit im Bauwesen relevanten Produktdatenmodelle weitergehende Untersuchungen notwendig sind.

Deswegen werden für die konzeptionelle Entwicklung die folgenden Entscheidungen getroffen:

- Die Literaturstudie hat ergeben, dass die in Abschnitt 3.2 vorgestellten semantischen Graphen für die Abbildung von MOF-M0 Instanzdaten beliebiger Produktdatenmodelle einen vielversprechenden Ansatz darstellen.
- Um die inkrementelle Änderung zwischen zwei Modellversionen zu ermitteln, wird ein Vergleich der Export-Resultate eingesetzt. Damit ist der erläuterte Ansatz unabhängig von den eingesetzten Autorensystemen und erhöht damit die Allgemeingültigkeit des Ansatzes. Gleichzeitig bedarf es keiner Interpretation von Datenrepräsentationen, die spezifisch für ein bestimmtes Autorensystem wären. Damit können weitreichende Parallelen zu etablierten Versionskontrollsystemen wie Git oder vergleichbaren Produkten erzielt werden.

Als wichtige Randbedingung wird darüber hinaus angenommen, dass ein Bauteil sowie andere Primärobjekte wie Räume oder logische Objekte über verschiedene Modellversionen hinweg ein eindeutig identifizierbares Merkmal aufweist (im Sinne einer GUID bzw. UUID). Diese Randbedingung erfüllen alle im Kontext dieser Arbeit betrachteten Datenmodelle.

Abb. 4.2 illustriert die verschiedenen technischen Ebenen, die im weiteren Verlauf berücksichtigt werden. Vorausgesetzt wird, dass die verwendeten Modellierungswerkzeuge in der Lage sind, die Ergebnisse von Entwurfs- und Planungsaufgaben in Form von BIM-Modellen abzubilden. Weiterhin wird angenommen, dass diese Softwareapplikationen über passende Schnittstellen verfügen, die erstellten (Disziplin-)Modelle in Repräsentationen zu exportieren, die den herstellerneutralen, offenen Datenmodellen folgen. Die einzelnen Modelle werden anschließend auf Objektbasis betrachtet, die der MOF-M0 Ebene entsprechen. Die Verwaltung einzelner Versionsinkremente erfolgt danach in einem lokalen Repository.

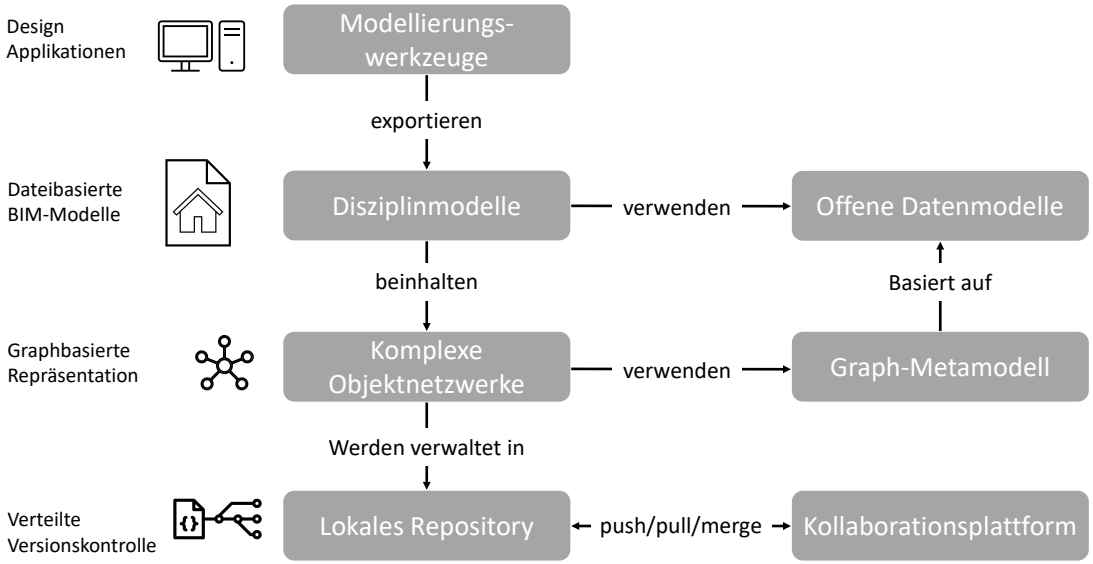


Abbildung 4.2: Technologische Ebenen der Versionskontrollmethodik

Zusätzlich wird die Triviallösung ausgeschlossen, die sich aus dem vollständigen Löschen der initialen Version und einem anschließenden, vollständigen Einfügen der aktualisierten Modellversion ergeben würde. Diese Lösung würde die derzeitige Situation des dateibasierten Modellaustauschs in keiner Weise verbessern. Die geschilderten Prinzipien zur textbasierten Versionskontrolle haben in Abschnitt 2.5.2 bereits verdeutlicht, dass es bei der Ermittlung eines Änderungsskriptes ausgehend von einem Ausgangs- und Endzustand in der Regel keine eindeutige, sondern mehrere Lösungen geben kann, die unterschiedlich effektiv sein können. Daher ist zu erwarten, dass dieses Phänomen ebenfalls für die Ableitung inkrementeller Änderungen zwischen den Graphrepräsentationen zweier Modelle auftreten wird. Für diesen Aspekt wird angenommen, dass die gewählte Beschreibung der vorgenommenen Änderungen in erster Linie verlustfrei erfolgen muss. Die Anwendung inkrementeller Verfahren muss nach erfolgreicher Übermittlung und Anwendung des abstrahierten Inkrements auf eine veraltete Version zwingend eine äquivalente Datenrepräsentation ergeben, die auch auf sendender Seite als aktualisiertes Modell vorlag. Diese Zielsetzung korreliert mit dem bereits eingeführten Begriff der *Transaktion*, der zuvor im Zuge der Konsistenzüberlegungen in Abschnitt 2.6 eingeführt wurde. Eine möglichst optimale Formulierung der Änderungsprozedur wird dabei nachrangig

betrachtet. Implikationen und weitere Optimierungspotenziale zu dieser Annahme werden später in Kapitel 7 diskutiert.

Abb. 4.3 visualisiert die einzelnen Schritte, die für die Ermittlung der Versionsinkremente notwendig sind und im Folgenden detailliert erklärt werden. Abschnitt 4.4 erläutert das Vorgehen zur Abbildung der MOF-M0 Instanzdaten auf die Graphstruktur sowie das zugrunde gelegte Graph-Metamodell. In Abschnitt 4.5 wird das Vorgehen zur Ermittlung des maximalen gemeinsamen Subgraphen beschrieben. Daraus leitet sich im Folgenden die Ermittlung gelöschter, modifizierter und hinzugefügter Informationen ab. Anschließend wird mit diesen Ergebnissen in Abschnitt 4.6 erörtert, wie aus der Kenntnis des Maximum Common Subgraphs passende Graphersetzungsregeln abgeleitet werden. Das Kapitel schließt mit der Anwendung der Graphersetzungsregel auf ein veraltetes Modell auf Empfängerseite, was in Abschnitt 4.7 dargestellt wird.

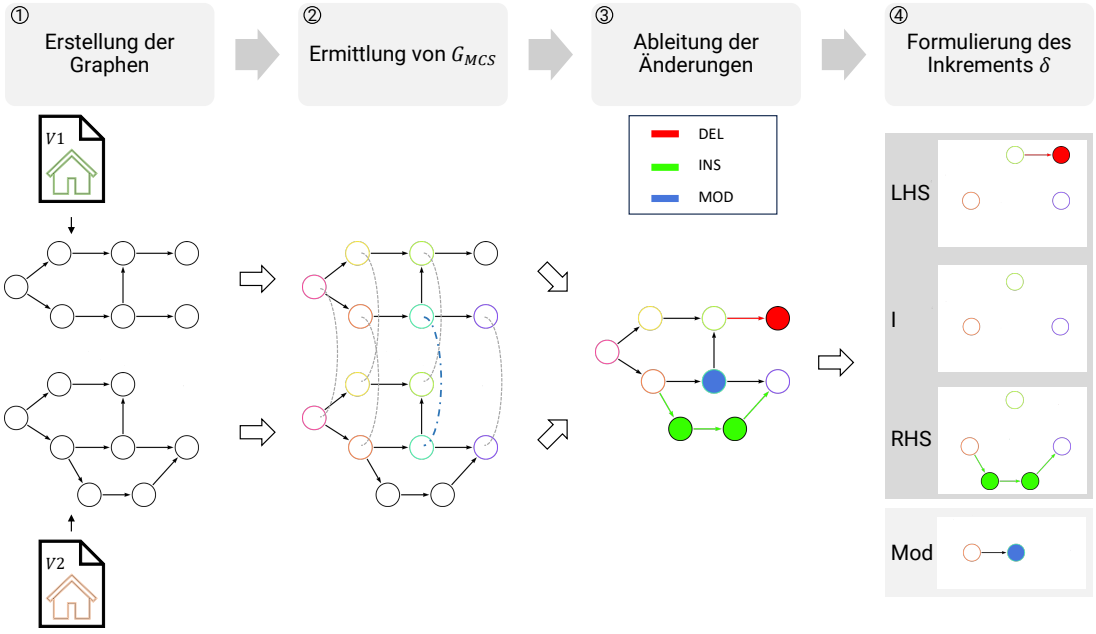


Abbildung 4.3: Vorgehen zur Ermittlung von Änderungen basierend auf zwei Versionen eines BIM-Modells

4.3 Ansatz zur Ermittlung eines Inkrements und dessen Austauschs zwischen einem Sender und einem Empfänger

Im Verlauf der weiteren Ausführungen werden verschiedene Begriffe genutzt, um Zustände, Zwischenergebnisse und Vorgänge zu bezeichnen. Abb. 4.4 gibt hierzu einen Überblick. Betrachtet wird der Datenaustausch zwischen einem Sender und einem Empfänger. Angenommen wird, dass beide Parteien über ein vollwertiges Dateimanagementsystem verfügen, in dem allgemeine Operationen für Dateien zur Verfügung stehen und die von allen gängigen Betriebssystemen bereitgestellt werden. Neben dem Dateisystem verfügen Sender und Empfänger über einen Graphspeicher (z.B. in Form einer lokal vorgehaltenen Graphdatenbank oder gleichartigem Speicher zur Interaktion mit Graphstrukturen), in

dem die Graphrepräsentationen der Modelle gespeichert werden und mit denen in den einzelnen Prozessschritten interagiert werden kann. Der Sender verfügt zu Beginn über zwei Versionen eines BIM-Modells, die in die Graphen G_{init} und G_{updt} übersetzt werden. Der Graph G_{init} beinhaltet die Informationen der initialen Modellversion, wohingegen der Graph G_{updt} jene des aktualisierten Modells enthält. Der Prozess zur Erstellung eines Inkrements wird als *Diff* bezeichnet. Auf empfangender Seite wird angenommen, dass ebenfalls die initiale Version des Modells und der korrespondierende Graph G_{init} vorliegt. Die Anwendung eines Inkrements auf einen Graph wird als *Patch* bezeichnet, woraus ein Ergebnisgraph G_{res} entsteht.

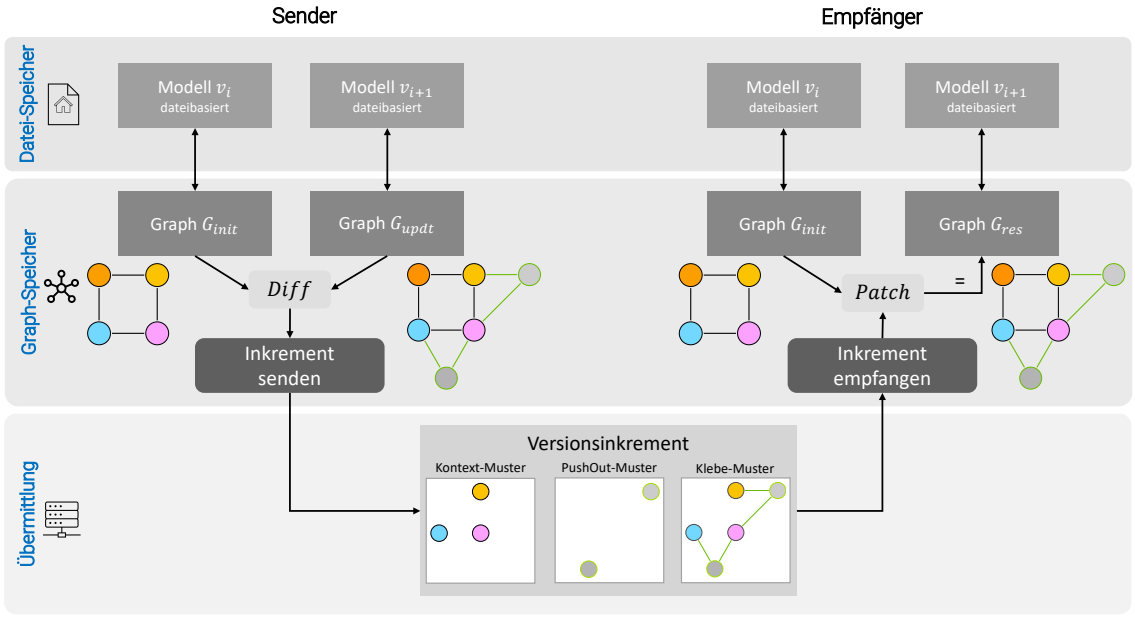


Abbildung 4.4: Systemarchitektur zur Versionskontrolle von BIM-Modellen auf Basis ihres Objektgefüges

4.4 Abbildung objektorientierter Produktdaten auf einen beschrifteten Eigenschaftsgraphen

Um die in Kapitel 2 beschriebenen Limitationen textbasierter Versionskontrollsysteme für die Verwaltung vernetzter Objektinformationen zu überwinden, wird zur Beschreibung eines MOF-M0 Modells ein LPG-Graph gewählt. Dieser kann die enthaltenen Informationen unabhängig von der gewählten Auszeichnungssprache oder spezifischen Serialisierungsstrategien abbilden. Die Formulierung des Graphen erfolgt dabei gemäß den von Hidders (2001) erläuterten Prinzipien. Die Gesamtarchitektur zur objektbasierten Versionskontrolle ist darüber hinaus unter anderem von den Arbeiten von Koch und Firmenich (2011), Nour et al. (2006) und Firmenich et al. (2005) inspiriert und durch moderne Konzepte aus dem Arbeiten mit der textbasierten Versionskontrollumgebung Git (Chacon, 2009) ergänzt.

Hierfür wird das folgende, allgemeine Graph-Metamodell verwendet, welches drei Arten von Knotentypen verwendet:

- **Primärknoten** bilden sämtliche Klasseninstanzen ab, die als Attribut ein eindeutiges, über Modellversionen hinweg konsistentes Identifikationsmerkmal aufweisen. Hierfür eignen sich beispielsweise Globally Unique Identifier (GUID).

- **Sekundärknoten** werden verwendet, um alle Instanzen innerhalb eines Modells im Graphen zu reflektieren, die über kein eindeutiges Merkmal verfügen, aber weitergehende Informationen mit dem Primärknoten über Assoziationen bereitstellen.

- **Beziehungsknoten** dienen der Abbildung von Viele-zu-Viele-Beziehungen. Zudem besitzen einzelne der in Abschnitt 2.3.5 untersuchten Datenmodelle sogenannte *objektifizierte Relationen*, die ebenfalls mit dieser Art von Knoten ausgedrückt werden.

Im Grundsatz wird jedes in der Instanzmenge enthaltene Objekt auf einen Knoten im Graph abgebildet. Es handelt sich daher um eine bijektive Abbildung der Instanzdaten auf die zugehörige Graphrepräsentation. Sämtliche Attribute, die an der jeweiligen Instanz gemäß zugehörigem Datenmodell vorzuhalten sind, werden als *Name-Wert-Paar* direkt an den Knoten als Eigenschaftensatz modelliert. Diesem wird zusätzlich ein *EntityType*-Attribut hinzugefügt, das den Namen der Klasse als textuellen Wert widerspiegelt. Sofern ein Objekt Assoziationen zu anderen Objekten besitzt, werden diese als Kanten zwischen den assoziierten Knoten abgebildet. Der Assoziationsname wird dabei in das Kantenattribut *relType* gespeichert. Handelt es sich dabei um eine Liste mehrerer Assoziationen, so wird ebenfalls die Position in der Liste in der Graphrepräsentation erhalten, indem ein weiteres Attribut mit Namen *listItem* hinzugefügt wird.

An dieser Stelle sei bereits darauf verwiesen, dass das gewählte Graph-Metamodell bewusst allgemein gehalten ist, um möglichst viele objektorientierte Strukturen abdecken zu können. Diese Flexibilität geht aber mit Nachteilen bei der konsistenten und datenmodellkonformen Transformation der Graphen einher, sodass die Graphen inkompatibel mit den zugehörigen Datenmodellen werden können. Abschnitt 7.2.4 erläutert die Vor- und Nachteile des gewählten Graph-Metamodells ausführlich und geht insbesondere auf mögliche Unzulänglichkeiten ein.

Um die verlustfreie Übersetzung der Instanzinformationen eines gegebenen Modells in seine Graphrepräsentation durchzuführen, muss noch die Zuordnung jeder Klasse des Datenmodells auf einen geeigneten Typen des Graph-Metamodells vorgenommen werden. Diese Abbildungsvorschrift muss für jedes Produktdatenmodell individuell festgelegt werden. Da die untersuchten Datenmodelle aber umfangreich vom Konzept der objektorientierten Vererbung Gebrauch machen, bietet es sich an, möglichst allgemeine Klassen des zugehörigen Datenmodells auf die jeweiligen Knotentypen abzubilden. Abb. 4.5 zeigt die Abbildung des IFC-Datenmodell auf das entworfene Graph-Metamodell.

Um alle Knoten zu identifizieren, die zu einer bestimmten Version des BIM-Modells gehören, wird jedem Knoten ein zusätzliches Label angefügt, das den Zeitstempel des Erstellungsdatums angibt. Typischerweise definiert dieses Datum, wann das BIM-Modell aus dem Autorenwerkzeug exportiert wurde. Auf diese Weise kann die Graphstruktur,

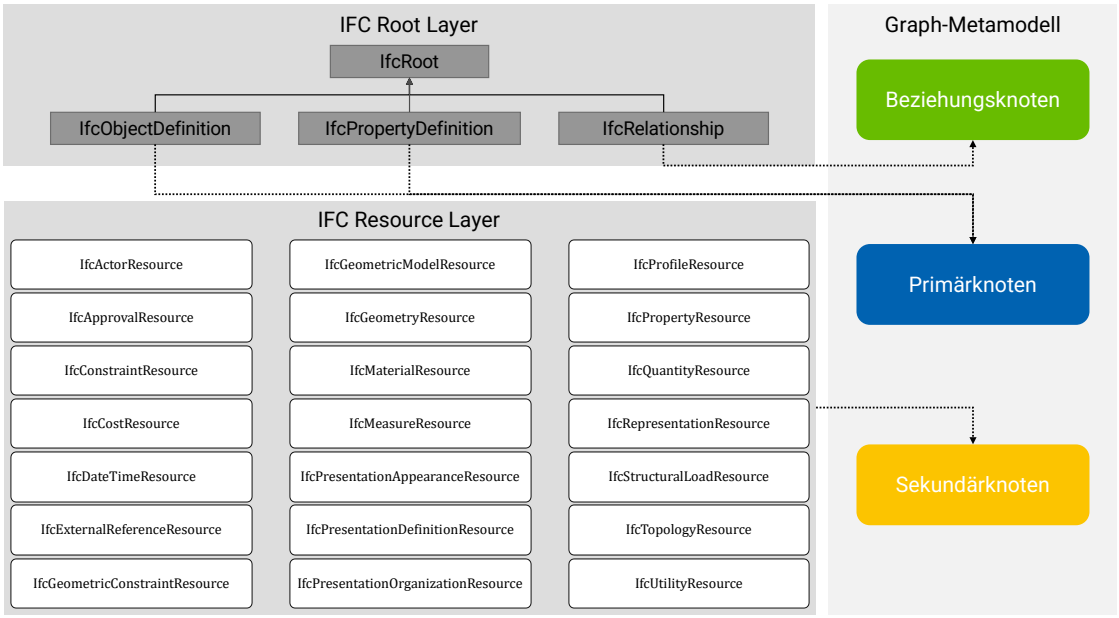


Abbildung 4.5: Gewählte Abbildungsvorschrift des IFC-Datenmodells auf das skizzierte Graph-Metamodell

die eine bestimmte Version eines Modells abbildet, leicht innerhalb des Graphspeichers identifiziert werden. Der Begriff der *Modellkomponente* wird darüber hinaus verwendet, um einen Subgraph zu beschreiben, der sich aus einem Primärknoten und mehreren Sekundärknoten zusammensetzt. Der Primärknoten repräsentiert dabei wesentliche Grundinformationen eines Bauteils, die unter anderem den Namen, eine Beschreibung und eindeutiges Merkmal umfassen. Die Sekundärknoten modellieren anschließend mit dem Bauteil assoziierte Informationen. Diese werden auch als *Ressourcen* bezeichnet und können dabei unterschiedliche Aspekte der gebauten Umwelt repräsentieren. Sie umfassen unter anderem Aspekte der geometrischen Bauteilrepräsentation, der Platzierung im Raum oder die Zuweisung von Materialeigenschaften. Der Begriff findet seinen Ursprung im IFC-Datenmodell [1].

Die Analyse verschiedener Datenmodelle hat gezeigt, dass im IFC-Datenmodell alle wesentlichen Herausforderungen zur Entwicklung einer vollwertigen Versionskontrollmethodik auf Objektbasis auftreten. Daher konzentrieren sich alle folgenden Erläuterungen vorrangig auf dieses Datenmodell. Genauer wird insbesondere die Serialisierung nach STEP-21 (ISO, 2016) eingegangen. Sollten Phänomene anderer Serialisierungsformate oder Datenmodelle für die Erläuterung ausgewählter Aspekte relevant sein, werden diese entsprechend hervorgehoben.

Exemplarisch wird nun folgende IFC-Datei und deren Wandlung in ihre Graphstruktur behandelt. Algorithmus 4.1 zeigt die nach STEP-P21 serialisierte Form.

[1] https://standards.buildingsmart.org/IFC/RELEASE/IFC4_3/HTML/chapter-8/index.html (Letzter Zugriff: 02.01.2024)

Algorithmus 4.1: IFC-Modell mit einer schlichten räumlichen Struktur und einer Modellkomponente

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [Ifc4NotAssigned]'),'2;1');
FILE_NAME('Cube_single.ifc',
  '2021-01-19T08:54:06',
  ('Sebastian Esser'),
  ('TUM'),
  'GeometryGymIFC v0.1.6.0 by Geometry Gym Pty Ltd',
  'UnitTestGenerator v1.0.0.0',
  'None');

FILE_SCHEMA (('IFC4'));
ENDSEC;

DATA;
#1 = IFCCARTESIANPOINT((0.0,0.0,0.0));
#2 = IFCAXIS2PLACEMENT3D(#1,$,$);
#3 = IFCLOCALPLACEMENT($,#2);
#4 = IFCSITE('2gG1du9OH4eQ4omNtwzfn1',$,'site',$,$,#3,$,$,$,$,$,$,$,$);
#5 = IFCPROJECT('2KrDsiaI1FkRwWUa5EvoNK',$,'GeomRep',$,$,$,$,(#10),$);
#6 = IFCRELAGGREGATES('1U7z3bi0Nu4L0UNA5m7S6j',$,$,$,#5,(#4));
#7 = IFCRECTANGLEPROFILEDEF(.AREA.,'rectangleProfileDef',$,4.0,6.0);
#8 = IFCEXTRUDEDAREASOLID(#7,$,#9,1.35);
#9 = IFCDIRECTION((0.0,0.0,1.0));
#10= IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,0.0001,#12,#13);
#11= IFCCARTESIANPOINT((0.0,0.0,0.0));
#12= IFCAXIS2PLACEMENT3D(#11,$,$);
#13= IFCDIRECTION((0.0,1.0));
#14= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',*,*,*,*,#10,$,.
    MODEL_VIEW.,$);
#15= IFCSHAPEREPRESENTATION(#14,'Body','SweptSolid',(#8));
#16= IFCPRODUCTDEFINITIONSHAPE($,$,(#15));
#17= IFCCARTESIANPOINT((2.0,5.0,1.0));
#18= IFCAXIS2PLACEMENT3D(#17,$,$);
#19= IFCLOCALPLACEMENT(#3,#18);
#20= IFCBUILDINGELEMENTPROXY('3xneIo5tr8TOYxqxH15Rkg',$,'Cuboid1',$,$,#19,
    #16,$,$);
#21= IFCRELCONTAINEDINSPATIALSTRUCTURE('3FNv6N_ur0zQ8tygS1XDuH',$,'Site','
    Site',(#20),#4);
ENDSEC;
END-ISO-10303-21;
```

Die nachfolgende Abb. 4.6 zeigt die Visualisierung des BIM-Modells in einem Modell-Viewer.

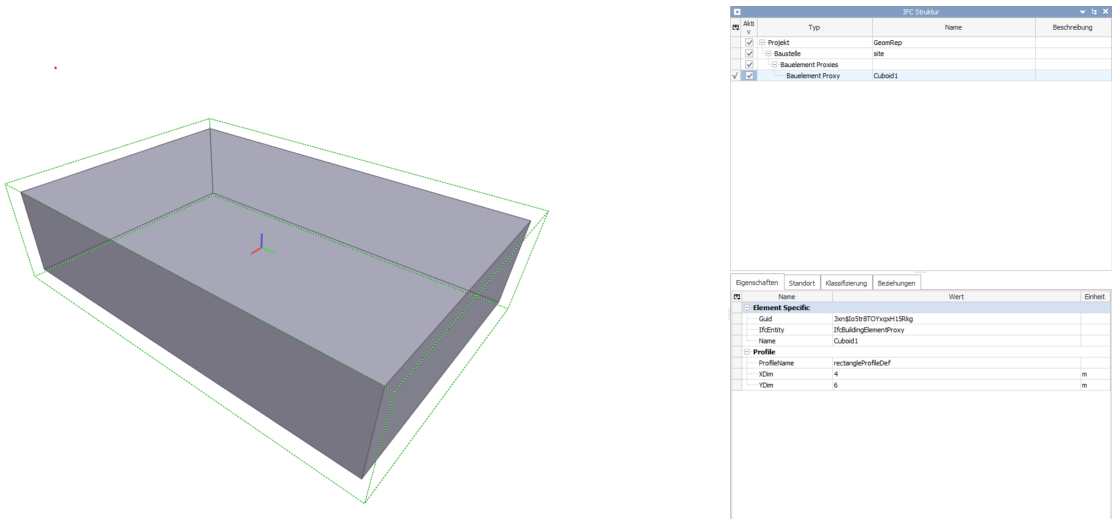


Abbildung 4.6: Visualisierung des BIM-Modells in einem Model-Viewer DataComp BIM Vision

Abb. 4.7 illustriert die zugrundeliegende vernetzte Objektstruktur als LPG-Graph. Jeder Knoten wird zum Zweck der folgenden Ausführungen mit der zugehörigen Entitätennummer der STEP-P21 Datei ausgestattet. Blau dargestellte Knoten illustrieren *Primärknoten*, grüne Knoten *Beziehungsknoten* und gelbe *Sekundärknoten*. Zur Vereinfachung werden in den Knotenattributen lediglich jene Attribute dargestellt, die einen Wert aufweisen und nicht Null sind (in SPF mit $ notiert). Bei den grau hinterlegten Feldern an den Kanten handelt es sich um die Assoziationsattribute, die zwei Instanzen in Beziehung zu einander setzen. Sofern es sich um eine Liste von Verweisen handelt, ist zudem die Listenposition im Attribut *listItem* angegeben. Dies ist beispielsweise bei den Kanten der Fall, die Knoten 21 mit 20 oder 16 mit 15 assoziieren. Wie beschrieben wird zusätzlich zu den im Datenmodell aufgeführten Attributen jedem Knoten das Attribut *EntityType* hinzugefügt, das die entsprechende Klasse des zugrundeliegenden Datenmodells spiegelt.

Aus graphtheoretischer Sicht handelt es sich bei der Repräsentation solcher Objektmodelle um gerichteten Graphen ohne Schleifen, deren Charakteristika in Abschnitt 3.1.4 eingeführt wurden. Zur Verbesserung der Lesbarkeit werden im Folgenden die Begriffe *Graph* und *Graphrepräsentation* synonym verwendet, um die im Modell enthaltenen Instanzen und deren Beziehungen in der Form eines LPG-Graphs zu bezeichnen.

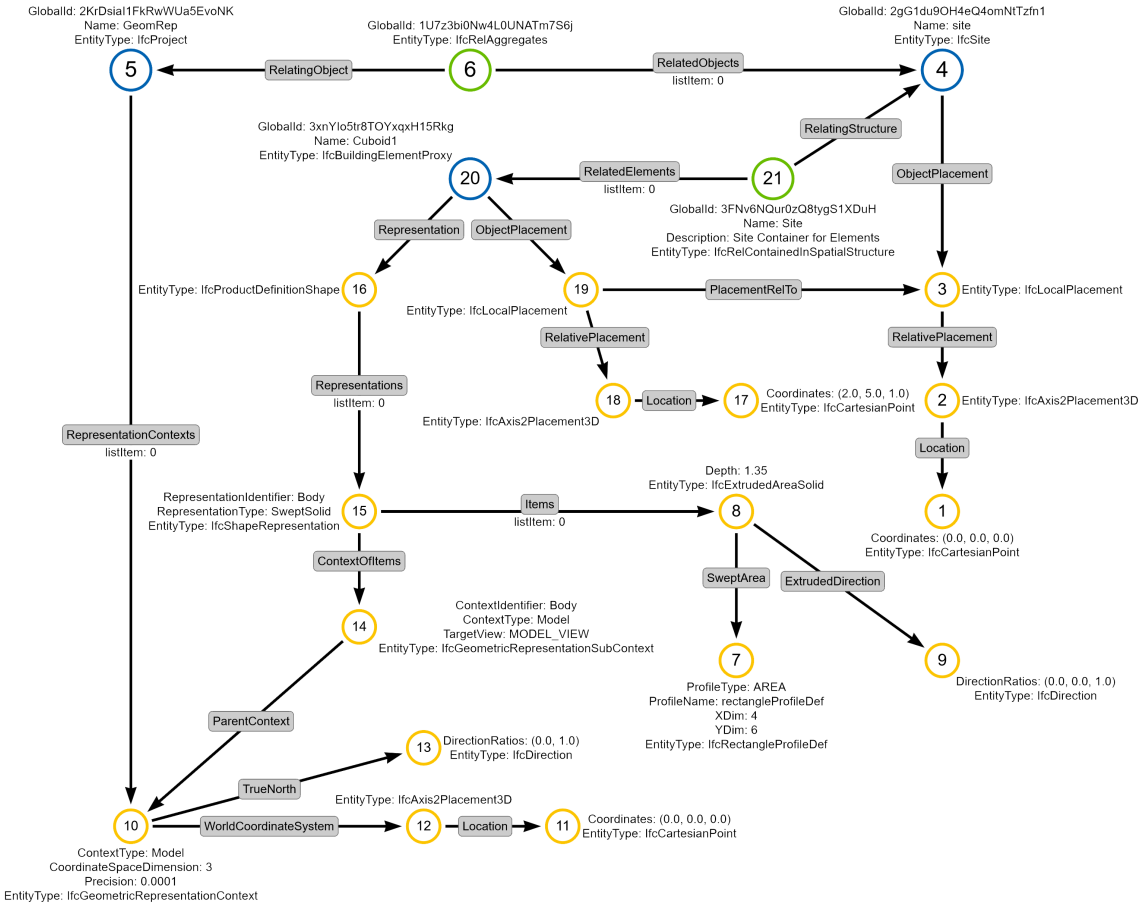


Abbildung 4.7: Graph G der in Algorithmus 4.1 serialisierten Modellinformationen

An diesem Beispiel lassen sich stellvertretend verschiedene Phänomene erläutern, die zuvor bereits in Abschnitt 3.2.3 dargelegt wurden. Für die mehrfache Passung von Mustern wird beispielhaft die Positionierung von Objekten im dreidimensionalen Raum herangezogen. Der kartesische Punkt, repräsentiert durch die IFC-Entität `IfcCartesianPoint` mit dem Koordinatenwert $(0.0, 0.0, 0.0)$, kann mit dem in der Cypher-Sprache formulierten Muster gesucht werden, dass Algorithmus 4.2 spezifiziert.

Algorithmus 4.2: Cypher-Statement zur Mustersuche aller kartesischen Punkte im Koordinatenursprung

```
MATCH p = (n{
    EntityType: "IfcCartesianPoint",
    Coordinates: "(0.0, 0.0, 0.0)"})
RETURN p
```

Als Ergebnis sind die beiden Knoten mit den Ziffern 1 und 11 zu erwarten, insgesamt also zwei Passungen für das spezifizierte Muster p im Graph G.

Die Tatsache, dass ein Muster mehrere Passungen in einem Graph haben kann, ist in vielen Situationen gewünscht. Für die Analyse von BIM-Modellen auf Basis ihrer Graphrepräsentationen kann dieser Umstand allerdings zusätzliche Betrachtungen notwendig machen, um Knoten und Kanten eindeutig in ihrer topologischen Lage adressieren zu können. Betrachtet wird beispielhaft eine Modifikation G' des zuvor erläuterten Graphs

G, bei dem der Knoten 11 entfernt wurde und die bisherige Kante zwischen Knoten 12 und 11 nun von Knoten 12 auf Knoten 1 verweist. Die übrigen Knoten und Kanten bleiben unverändert. Der modifizierte Graph G' ist in Abb. 4.8 dargestellt. Die modifizierte Kante ist in blau hervorgehoben. Das durch den Graph G' repräsentierte BIM-Modell bleibt in seinem ausgewerteten Zustand gleich, da lediglich eine redundant vorliegende Information entfernt wurde. Gleichzeitig ist festzuhalten, dass sich die Graphen G und G' in den Knoten- und Kantenmengen unterscheiden.

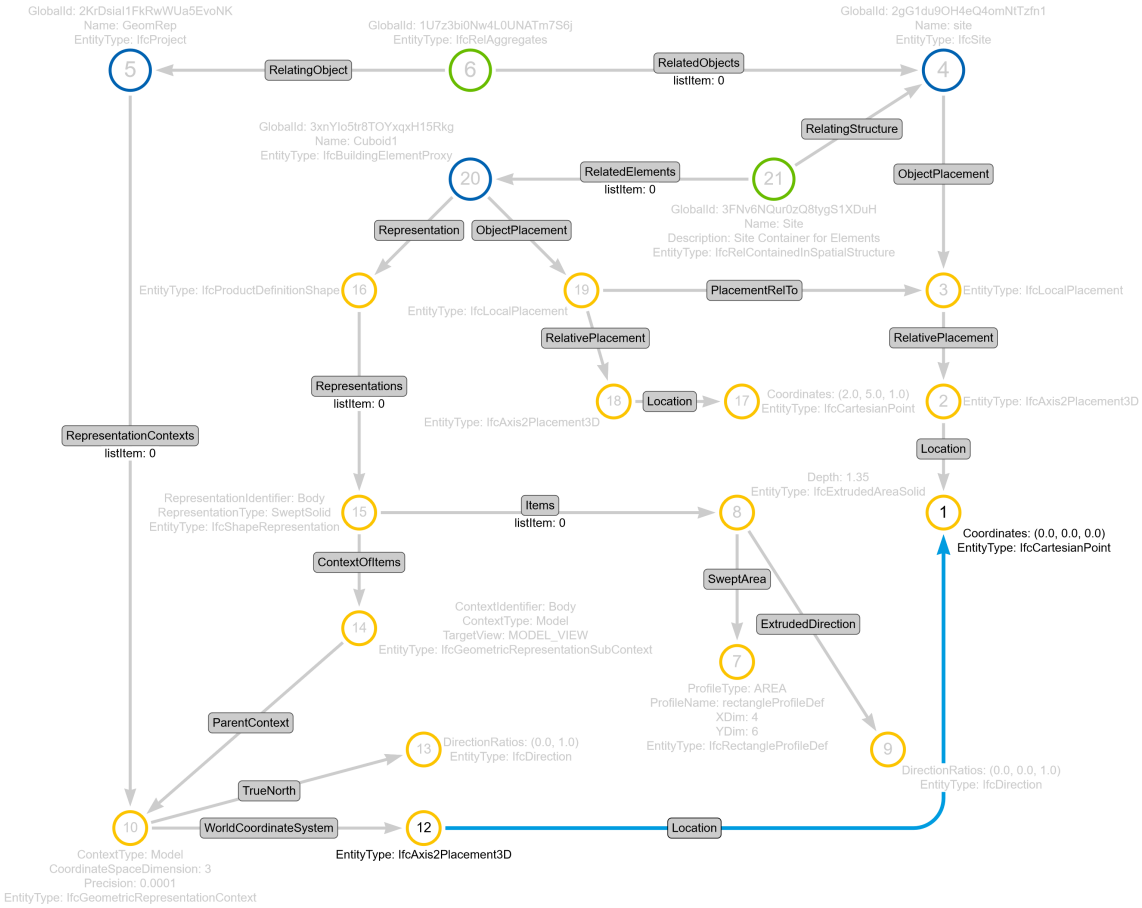


Abbildung 4.8: Modifizierter Graph G' der in Algorithmus 4.1 serialisierten Modellinformationen

Die zuvor erläuterte Mustersuche nach einem kartesischen Punkt im Koordinatenursprung würde in diesem Falle nur noch eine Passung in Form des Knotens 1 ergeben. Ebenso lässt sich festhalten, dass der modifizierte Graph G' homomorph zu G ist, also sämtliche Nachbarschaften zwischen den an Knoten gespeicherten Informationen beibehält. Die Knotenmenge V' ist allerdings durch das Löschen des Knotens 11 um ein Element verkleinert worden. Daraus folgt, dass die für eine bijektive Abbildung der Knotenmengen zwischen V und V' notwendige gleiche Mengengröße nicht mehr gegeben ist. Demnach kann zwischen den beiden Graphen keine isomorphe Abbildung spezifiziert werden.

Die zu Beginn des Kapitels eingeführten Abwägungen müssen an dieser Stelle nochmals weitergehend behandelt werden. Das erläuterte Beispiel in Abb. 4.7 und Abb. 4.8 verdeutlicht, dass es eine Unterscheidung zwischen *graph-strukturellen* Aspekten und der

Interpretation der in den Graphen gespeicherten Informationen bedarf. Aus der *interpretierten* Sichtweise beinhalten beide Graphrepräsentationen G und G' die exakt gleiche planerische Information. Selbst zahlreiche Softwareapplikationen, die Funktionen zum Modellvergleich anbieten, werden beide Modelle nach dem Import in die Programmumgebung als äquivalent ansehen. Aus *graph-struktureller* Sicht handelt es sich bei der vorgenommenen Änderung zwischen den Graphen G und G' allerdings sehr wohl um eine relevante Änderung, die als solche entsprechender Betrachtung bedarf. Im illustrierten Beispiel wurde eine *Normalisierung* (siehe dazu auch Abschnitt 2.6) der Graphstruktur vorgenommen und die in den Knoten 1 und 11 gegebene Informationsdoppelung entfernt. Dennoch stellen beide Graphen *ausgewertet* eine äquivalente Information für den Nutzer bereit. Sie unterscheiden sich aber in ihrer zugrundeliegenden Struktur aus mengen- und graphentheoretischer Sicht. Für den weiteren Verlauf der Arbeit werden Überlegungen, ob zwei Graphen G und G' zur gleichen *Auswertung* beziehungsweise *Interpretation* führen können, nicht weiter beleuchtet. Vielmehr werden auch Änderungsoperationen auf graphstruktureller Ebene, die möglicherweise keine Auswirkung auf die Interpretation des Modells hätten, als zu erfassende Änderungen angesehen, die angemessen ausgetauscht werden müssen. Sofern die beschriebenen Mechanismen zur formalen Beschreibung der Änderungen auf graph-struktureller Sicht Auswirkungen auf deren ingenieurmäßige Interpretation erwarten lassen, werden diese an den geeigneten Stellen erläutert. Darüber hinaus fokussieren sich die folgenden Ausführungen auf die Betrachtung der Graphen ohne Auswertung des zugrundeliegenden Ingenieurwissens.

4.5 Ermittlung eines gemeinsamen Subgraphs zwischen zwei Modellversionen

Im vorangegangenen Abschnitt wurde dargelegt, wie Modellinformationen in die gewählte Graphrepräsentation überführt werden. Um nun die inkrementelle Änderung zwischen zwei Modellversionen zu ermitteln, müssen die zwei zu betrachtenden Modellversionen in den Graph-Speicher überführt werden. Die Graphen der Versionen werden im folgenden mit G_{init} für den Initialzustand und G_{updt} für den modifizierten Zustand bezeichnet.

Gesucht wird nun ein maximaler gemeinsamer Teilgraph (im Englischen *Maximum Common Subgraph*) G_{MCS} zwischen den Graphen beider Modellversionen auf sendender Seite. Der Graph G_{MCS} ist ein isomorpher Teilgraph zu beiden Graphen G_{init} und G_{updt} (Bunke, 1997; McCreesh et al., 2017). Dieser Teilgraph repräsentiert dabei anschaulich betrachtet alle Bestandteile des Modells, die im Übergang zwischen den beiden Versionen nicht verändert wurden. Weiter kann spezifiziert werden, dass es sich bei dem zu ermittelnden Teilgraph um einen *induzierten* Teilgraph handeln soll.

Abb. 4.9 illustriert die Fragestellung. Die gewünschten isomorphen Abbildungen sind mit φ_1 und φ_2 bezeichnet.

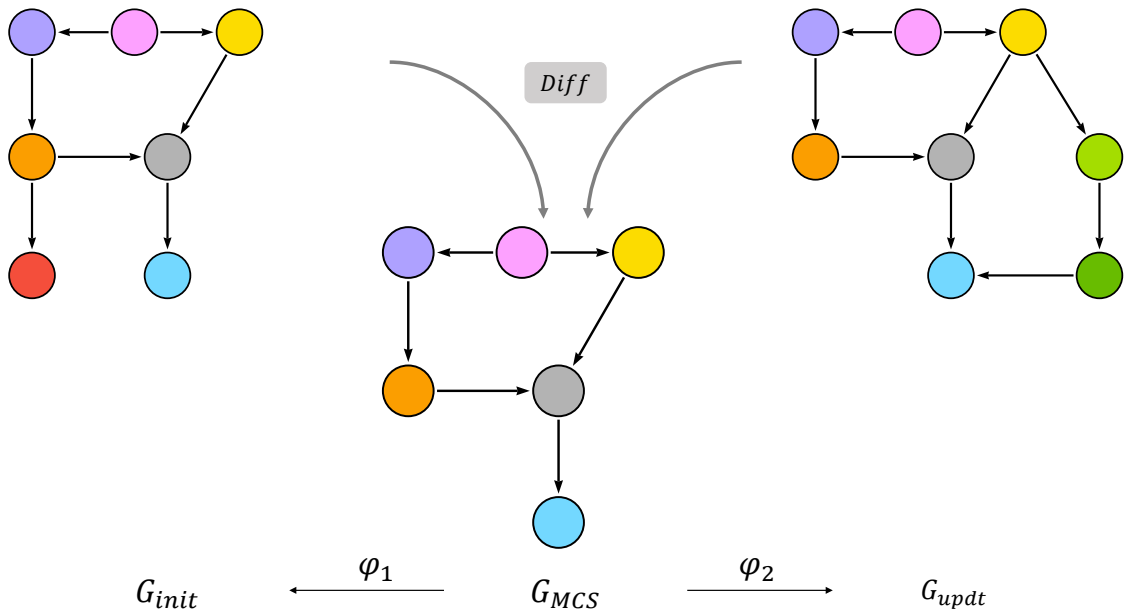


Abbildung 4.9: Schematische Darstellung des zu ermittelnden maximalen gemeinsamen Subgraphs G_{MCS} ausgehend von den Graphen G_{init} und G_{updt}

Zur Ermittlung dieses Teilgraphs gibt es wie in Abschnitt 3.1.1 erwähnt bisher keine allgemeingültige Lösung, die für beliebige Graphen anwendbar ist. Wie dargestellt gibt es allerdings diverse (deterministische) Verfahren zur Bestimmung der maximal übereinstimmenden Subgraphen, die auf der Tiefensuche basieren. Daher bedient sich auch die im Folgenden beschriebene Methodik einer rekursiven Tiefensuche. Zusätzlich ist festzuhalten, dass es sich bei den untersuchten Datenmodellen allesamt um DAG-Repräsentationen handelt. Somit ist es möglich, alle Graphrepräsentationen der Modelle der *topologischen Sortierung* zu unterziehen. Diese Eigenschaft wird später verwendet, um Sekundärknoten ohne eindeutigen Merkmalen in Mustern derart zu beschreiben, dass sie in beliebigen Versionen des Graphs eindeutig gefunden werden.

Um die Tiefensuche zum Vergleich beider Versionen zu initialisieren, bedarf es eines Startkriteriums, das für das anfängliche Annähern beider Graphen benötigt wird. Genutzt wird hierfür eine Charakteristik des in Abschnitt 4.4 eingeführten Graph-Metamodells. Dort wurde definiert, dass Instanzen, die als *Primärknoten* repräsentiert werden, über ein eindeutiges Merkmal verfügen müssen, welches über mehrere Modellversionen hinweg als persistent zu erwarten ist. Diese Knoten eignen sich daher ideal für eine initiale Ausrichtung beider Graphen und als Startpunkt für eine rekursive Tiefensuche. Im Falle des IFC-Datenmodells eignen sich als Startkombination die Instanzen der Entität `IfcProject`. Gemäß Dokumentation ist zu erwarten, dass hiervon in einem Modell lediglich eine Instanz enthalten ist [2].

Von dort aus wird die Tiefen-Traversierung gestartet. In jedem Traversierungsschritt werden sämtliche Knoten behandelt, die direkt adjazent zum untersuchten Knotenpaar sind und

[2] https://standards.buildingsmart.org/IFC/RELEASE/IFC4_3/HTML/lexical/IfcProject.htm Letzter Zugriff: 28. September 2023

mit einer gerichteten, den unprozessierten Knoten zugewandten Kante verbunden sind. Als äquivalent werden Knoten aus dem Graph G_{init} und G_{updt} dann betrachtet, wenn die folgenden Bedingungen eingehalten sind:

- Die Knoten sind an die zuvor gekoppelten Knoten mit dem gleichen *RelType* angeschlossen.
- Die adjazenten Knoten weisen den gleichen Attributwert im Attribut *EntityType* auf.
- Sofern es sich um eine Aggregation mehrerer Entitäten unter dem gleichen *RelType* handelt, wird zusätzlich die im Attribut *ListItem* gespeicherte Position berücksichtigt.
- Einer oder beide Knoten dürfen nicht bereits in vorherigen unterschiedlichen Äquivalenzpaaren inkludiert worden sein (eine genaue Erläuterung folgt in Abschnitt 4.10).

Sind zwei Knoten als äquivalent eingestuft, wird eine Kante des Typs *equivalent_to* eingefügt und so die beiden Graphen sukzessive mit einander verknüpft. Zusätzlich werden die Knoten sowie die zugehörige Kante, entlang derer in diesem Schritt traversiert wurde, in den gemeinsamen Subgraph G_{MCS} hinzugefügt. Abb. 4.10 veranschaulicht den Abgleich der direkt adjazenten Knoten zu bereits als äquivalent eingestuften Knotenpaaren.

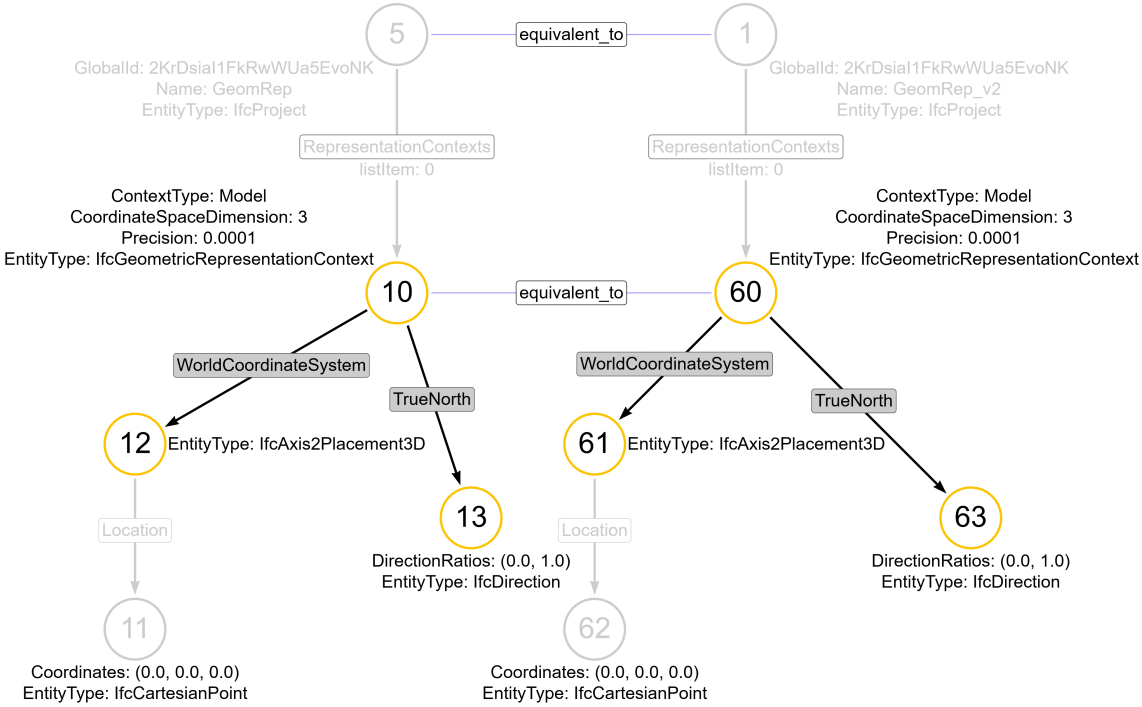


Abbildung 4.10: Vergleich der direkt adjazenten Knotenpaare zu einem bereits als äquivalent eingestuften Knotenpaar

Betrachtet werden im aktuellen Rekursionsschritt die adjazenten Knoten, die eine gerichtete Kante ausgehend von den Knoten $v_{init} = 10$ und $v_{updt} = 60$ haben. Dies sind die Knoten 12 und 13 als Teil von G_{init} sowie die Knoten 61 und 63 aus G_{updt}. Basierend auf dem Kantenattribut *RelType* stellen die Knotenpaare $12 - 61$ und $13 - 63$ gemäß den zuvor eingeführten Kriterien äquivalente Knotenpaare dar.

Sind auf dieser Basis zwei Knoten als äquivalent eingestuft, werden anschließend die Attribute beider Knoten miteinander verglichen. Sofern Abweichungen erkannt werden, werden diese als *graph-semantische Modifikationen* festgehalten. Im illustrierten Beispiel weisen beide Knotenpaare $12 - 61$ sowie $13 - 63$ übereinstimmende Knotenattribute auf, sodass keine Modifikation festzuhalten ist. Sollte sich der Wert eines Attributes verändert haben (beispielsweise der Wert des Attributs $Name$ am Knotenpaar $5 - 1$), so wird eine entsprechende *semantische Modifikation* erkannt und gespeichert.

Sofern in einem der beiden untersuchten Teilgraphen weitere Knoten vorhanden sind, werden diese als *topologische* oder *graph-strukturelle Modifikationen* erfasst. Sind Knoten in G_{init} vorhanden, zu denen es kein Gegenstück in G_{updt} gibt, wurden Informationen im Versionsschritt entfernt. Umgekehrt stellen Knoten, die nur in G_{updt} aber nicht in G_{init} vorhanden sind, neu hinzugefügte Informationen dar. Deren detaillierte Aufbereitung als Graphtransformation wird später in Abschnitt 4.6 eingeführt.

Besonderes Augenmerk muss nun auf jene Knoten gelegt werden, die Beziehungsknoten zwischen mehreren Objekten etablieren. Aus der Analyse verschiedener gängiger Softwareprodukte ging hervor, dass die Vergabe von *global* eindeutigen Merkmalen für diese Objekte sehr unterschiedlich gehandhabt wird. Um die Robustheit des Ansatzes zu erhöhen, wurde daher entschieden, diese Knoten erst nach Abschluss der Prozessierung aller anderen Modellkomponenten zu untersuchen und für die Verarbeitung der Beziehungsknoten ergänzende Ansätze zu entwickeln. Für die Äquivalenz werden hier im Gegensatz zu den Vorgehen für Primär- und Sekundärknoten nicht nur spezifische Attribute am Knoten selbst betrachtet, sondern auch die Nachbarschaftsbeziehungen. Als äquivalent werden Beziehungsknoten in den folgenden Fällen betrachtet:

- Sofern ein eindeutiges Merkmal an den Beziehungsknoten vorhanden ist und zwei Knoten $v_1 \in V_{init}, v_2 \in V_{updt}$ den gleichen Wert für dieses Attribut aufweisen.
- Sofern alle unveränderten Knoten, die direkt adjazent zu den zwei untersuchten Beziehungsknoten sind, als äquivalent eingestuft wurden.

In Abb. 4.11 wird der letztgenannte Fall illustriert. Angenommen wird, dass die Beziehungsknoten $1 \in G_{init}$ und $2 \in G_{updt}$ über kein eindeutiges Attribut verfügen oder deren Attributwerte Unterschiede aufweisen. Da die Knoten möglicherweise dennoch gleiche Elemente in Relation setzen, werden die adjazenten Knoten betrachtet, die unverändert und damit Teil des gemeinsamen Teilgraphs G_{MCS} sind. Diese sind durch die Anwesenheit einer Äquivalenzkante identifizierbar. Die Beziehungsknoten werden als äquivalent eingestuft, wenn zu allen adjazenten Knoten ein entsprechendes Gegenstück im jeweils anderen Graph existiert.

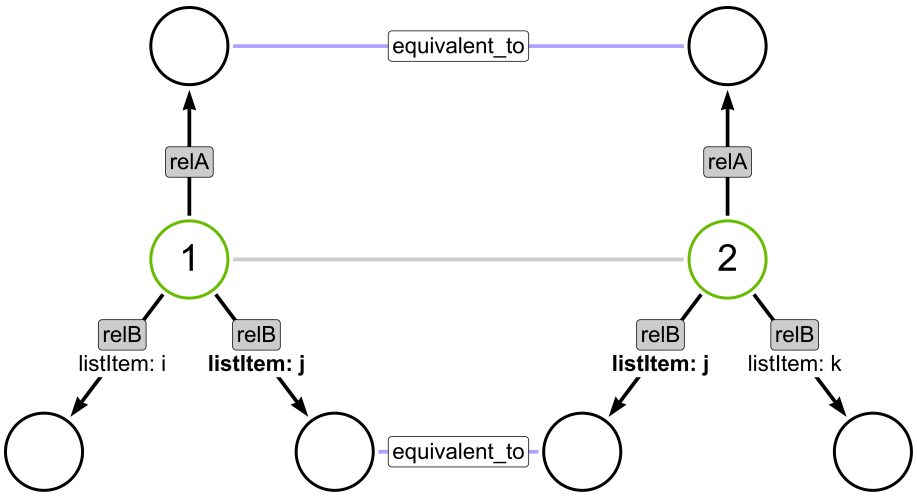


Abbildung 4.11: Untersuchung zweier Beziehungsknoten hinsichtlich ihrer Äquivalenz aufgrund ihrer adjazenten Knoten

Hierfür ist wie folgt vorzugehen:

1. Ermittle alle Beziehungsknoten $c_{init} \in G_{init}$.
2. Für jeden Knoten c_{init}:
   - Ermittle das Muster p_{init}, das aus dem Beziehungsknoten c_{init} und allen adjazenten Knoten besteht, die Teil des gemeinsamen Teilgraphs G_{MCS} sind.
   - Prüfe, ob das Muster auch im Graph G_{updt} vorhanden ist, indem das aus G_{init} abgeleitete Muster ohne den eindeutigen Merkmalen im Graph G_{updt} gesucht wird.
   - Wird im vorherigen Schritt eine Passung erzielt, wiederhole das Verfahren mit dem Knoten $c_{updt} \in G_{updt}$, der Teil der Passung des Musters p_{init} in G_{updt} ist.
   - Wird eine Passung für p_{updt} in G_{init} erzielt, können die Knoten c_{init} und c_{updt} als äquivalent eingestuft werden. Andernfalls sind entsprechende topologische Modifikationen abzuleiten.

Bei dem beschriebenen Verfahren handelt es sich um einen rekursiven Algorithmus. Sobald zwei Knotenpaare als äquivalent eingestuft sind, werden die hierzu direkt adjazenten Knoten untersucht, zu denen gerichtete Kanten ausgehend von dem bereits untersuchten Knotenpaar existieren. Als Abbruchkriterien dieser Tiefensuche werden folgende Bedingungen definiert:

- Ein Knoten im untersuchten Knotenpaar hat keine ausgehenden Kanten und damit keine weiteren adjazenten Knoten (End-Knoten-Fall)
- Ein oder beide Knoten eines Knotenpaares $v_1 \in V_{init}, v_2 \in V_{updt}$ wurden bereits in einem vorherigen Schritt untersucht und als äquivalent eingestuft.

Für letzteren Fall ist es notwendig, dass die Reihenfolge, in der alle Knoten traversiert werden, zusätzlich zur ermittelten Äquivalenz gespeichert wird. Als Vorteil ergibt sich daraus, dass die topologische Lage von Sekundärknoten anschließend durch einen eindeutigen Pfad beschrieben werden kann, indem die Teilsequenz zwischen dem gewünschten (Sekundär-)Knoten und dem zuvor auftretenden Primärknoten ermittelt werden kann. Wie bereits zuvor gezeigt, können Muster ohne eindeutigen semantischen Merkmalen zu mehrfachen Passungen in den behandelten Graphrepräsentationen führen. Insbesondere für die Inkremente, die lediglich eine Änderung von Attributen eines Knotens bewirken, ist das Auffinden der zu modifizierenden (Sekundär-)Knoten besonders wichtig.

Beispielhaft sei im Folgenden die Änderung der Positionierung einer Modellkomponente skizziert [3]. Die Position eines Bauteils beschreibt sich in Abhängigkeit zu dem Koordinatensystem des gesamten Projekts. Die entsprechende Koordinate spiegelt sich wiederum in einer Instanz der Entität `IfcCartesianPoint` wider. In Abb. 4.7 wird diese Information für die Modellkomponente $Cubiod1$ durch den Knoten 17 ausgedrückt und über den Pfad $20 \rightarrow 19 \rightarrow 18 \rightarrow 17$ mit dem Primärknoten verknüpft, der wiederum den zugehörigen Primärknoten der Modellkomponente bildet. Die Referenz auf das übergeordnete Koordinatensystem des Projektes ist in Form der Kante *PlacementRelTo* zwischen den Knoten 19 und 3 gegeben. Um dem in Algorithmus 4.2 erläuterten Problem passend zu begegnen, wird jede semantische Änderung immer über einen Pfad beschrieben, der den Primärknoten der betroffenen Modellkomponente beinhaltet. Zur Adressierung der Position der Modellkomponente würde das eindeutige Muster zum Auffinden der zur Modellkomponente gehörigen Koordinate daher lauten:

Algorithmus 4.3: Cypher-Statement zur eindeutigen Mustersuche der Positionierung einer IFC-Modellkomponente

```
MATCH p =
(A:PrimaryNode{
    EntityType:"IfcBuildingElementProxy",
    GlobalId:"3xnYIo5tr8TOYxqxH15Rkg"})
-[a:rel{relType:"ObjectPlacement"}]->
(B:SecondaryNode{EntityType:"IfcLocalPlacement"})
-[b:rel{relType:"RelativePlacement"}]->
(C:SecondaryNode{EntityType:"IfcAxis2Placement3D"})
-[c:rel{relType:"Location"}]->
(D:SecondaryNode{EntityType:"IfcCartesianPoint"})
```

4.6 Ableitung gelöschter, modifizierter und hinzugefügter Graphbestandteile und Formulierung des Inkrements

Im vorherigen Schritt wurde ein gemeinsamer Teilgraph G_{MCS} ermittelt, der alle unveränderten Knoten und Kanten zwischen den beiden untersuchten Versionen beschreibt.

[3] Weitergehende Informationen zum Umgang mit Positionierungen sind in der IFC-Dokumentation zu finden: https://standards.buildingsmart.org/IFC/RELEASE/IFC4_3/HTML/concepts/Product_Shape/Product_Placement/Product_Local_Placement/content.html (zuletzt geöffnet am 02.11.2023)

Gleichzeitig wurden bereits topologische und semantische Unterschiede detektiert, wenn zu einem Knotenpaar adjazente Knoten gefunden wurden, zu denen kein passendes Gegenstück existierte, oder sich Änderungen in den Attributen ergeben haben.

Abb. 4.12 zeigt das Unified Modeling Language (UML)-Diagramm mit allen Klassen, die zur Beschreibung eines Versionsinkrements herangezogen werden.

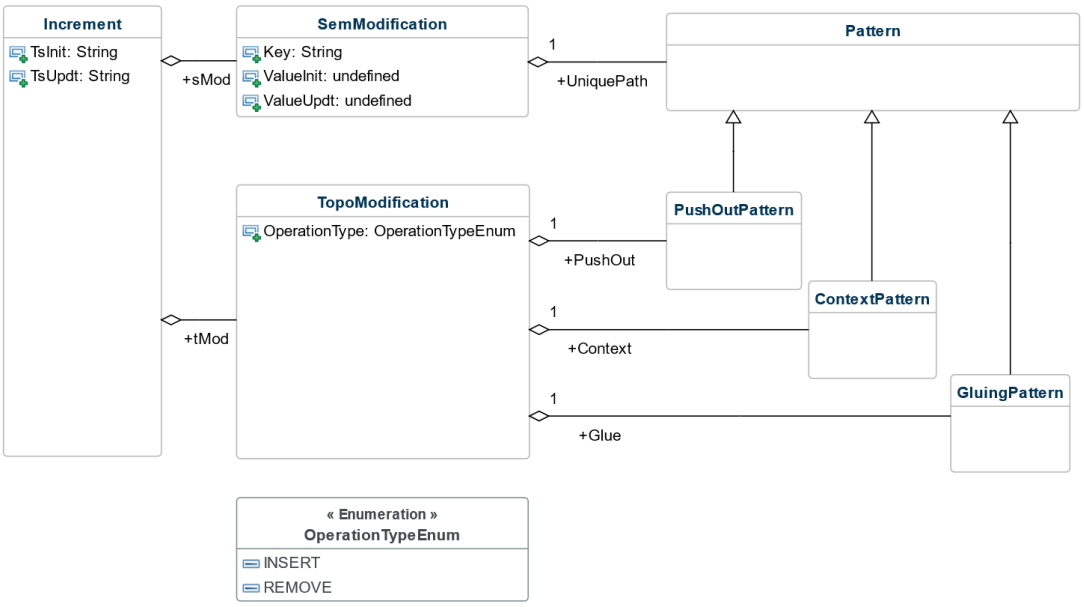


Abbildung 4.12: UML-Diagramm zur Beschreibung eines Versionsinkrements

Grundsätzlich setzt sich ein Versionsinkrement aus semantischen und topologischen Modifikationen zusammen, deren Charakteristiken jeweils als eigene Klassen modelliert werden. Eine topologische Modifikation kann dabei entweder das Entfernen oder das Hinzufügen eines Teilgraphs beschreiben und durch das Attribut *OperationType* in der jeweiligen Instanz der Klasse *TopoModification* ausgedrückt werden. Die Ermittlung und vollständige Beschreibung semantischer Modifikationen wird in Abschnitt 4.6.1 erläutert, wohingegen topologische Änderungen umfangreicherer Analysen bedürfen. Diese werden in Abschnitt 4.6.2 erläutert.

4.6.1 Erfassung semantischer Modifikationen

Im Falle semantischer Modifikationen wurden im vorgelagerten Schritt Änderungen detektiert, bei denen sich einzelne Knotenattribute in ihrem Wert verändern, der entsprechende Knoten aber in beiden Versionen an der gleichen topologischen Position im Graph aufgefunden wurde. Folglich reduziert sich die notwendige Transformation auf die Spezifikation eines Musters, um den zu modifizierenden Knoten eindeutig im Graph zu finden, und das anschließende Modifizieren des betroffenen Attributs von seinem initialen zu seinem aktualisierten Wert durchzuführen. Wie in Abschnitt 3.2.3 dargelegt, ist es insbesondere für Modifikationen von Sekundärknoten notwendig, ein Muster zu definieren, welches den betreffenden Knoten exakt im Graph findet. Hierfür wird die topologische Sortierung

herangezogen, die zuvor bereits für die Gruppe der DAG-Graphen in Abschnitt 3.1.4 erläutert wurde. Durch diese Sortierung kann ermittelt werden, von welchem Primärknoten aus der von der Modifikation betroffene Knoten das erste Mal aufgesucht wurde. Unter Einbeziehung des Primärknotens mit seinem eindeutigen Attribut wird sichergestellt, dass das spezifizierte Muster exakt eine Passung im zu aktualisierenden Graph haben wird. Aus graphentheoretischer Sicht kann das Muster weitergehend auf den Begriff des *Pfads* eingeschränkt werden, da jede semantische Modifikation exakt eine Attributveränderung behandelt und somit lediglich die Lage eines einzelnen Knotens im Graph eindeutig beschrieben werden muss.

Da alle Knoten, die zu jenem Knoten mit semantischer Änderung führen, Teil des gemeinsamen Teilgraphs sind, genügt es, das Muster ohne ausführliche Spezifikation aller Knotenattribute für jeden einzelnen Knoten zu beschreiben. Stattdessen wird für jeden Knoten lediglich der Entitätentyp sowie für jede Kante der Relationstyp spezifiziert. Sofern eine Sammlung mehrerer Zeiger im Muster zu behandeln ist, muss zusätzlich das Attribut *listItem* an der entsprechenden Kante in das Muster aufgenommen werden. Das eindeutige Muster ergibt sich damit für den allgemeinen Fall wie in Abb. 4.13 gezeigt. Diese Vereinfachung der Muster führt außerdem zu der Möglichkeit, semantische Modifikationen in beliebiger Reihenfolge auf eine überholte Version des Graphs anzuwenden.

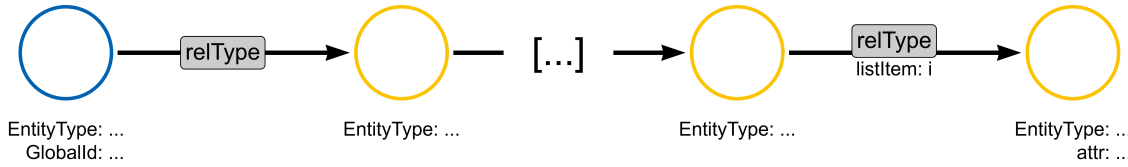


Abbildung 4.13: Allgemeine Beschreibung zur eindeutigen Passung eines beliebigen Knotens durch Zuhilfenahme eines Primärknotens

Sofern mehrere Knoten semantische Änderungen erfahren haben, können die zugehörigen Pfade auch in ein gesamtheitliches Muster zusammengefasst werden. Für die Anwendung dieser Modifkationen auf die überholte Graphrepräsentation wird dann zuerst eine Passung des Musters gesucht und anschließend alle geänderten Attribute entsprechend mit dem *SET*-Befehl modifiziert.

Ein anschauliches Beispiel für eine rein semantische Transformation ist in Abschnitt 4.9.1 gegeben. Zudem wird eine umfangreichere Anwendung in Abschnitt 6.2.1 erläutert, die insbesondere die aggregierte Anwendung mehrerer semantischer Modifikationen beinhaltet.

4.6.2 Erfassung topologischer Modifikationen

Etwas komplexer gestaltet sich die Ermittlung topologischer Modifikationen. Für die detektierten topologischen Änderungen bedarf es einer umfangreicheren Analyse der vorgenommenen Modifikationen als bei semantischen Veränderungen, bei denen bereits mit der Ermittlung des maximalen gemeinsamen Teilgraphs G_{MCS} alle notwendigen Informationen zur umfassenden Beschreibung vorliegen. Während des Diff-Prozesses können

keine direkten Aussagen über Knoten beziehungsweise Teilgraphen abgeleitet werden, die sich an die als gelöscht und hinzugefügt erkannten Knoten anschließen. Abb. 4.14 illustriert das Problem schematisch. In Rot sind Knoten und Kanten des initialen Graphs illustriert, zu denen kein Gegenstück im aktualisierten Graph existieren. In Grün sind neu hinzugefügte Knoten und Kanten dargestellt, die nur im aktualisierten Graph enthalten sind. Vom Diff-Algorithmus wird aufgrund der lokalen Betrachtung der direkten Kindknoten nur festgestellt, dass Knoten 2 mit Kante a kein Gegenstück in G_{updt} besitzt. Gleiches gilt für Knoten 5 und Kante d, die wiederum keine Entsprechungen im initialen Graph G_{init} aufweisen. Daher müssen im weiteren Verlauf alle entfernten und eingefügten Teilgraphen sowie ihre verbindenden Kanten zum unveränderten Teil des Graphs ermittelt werden.

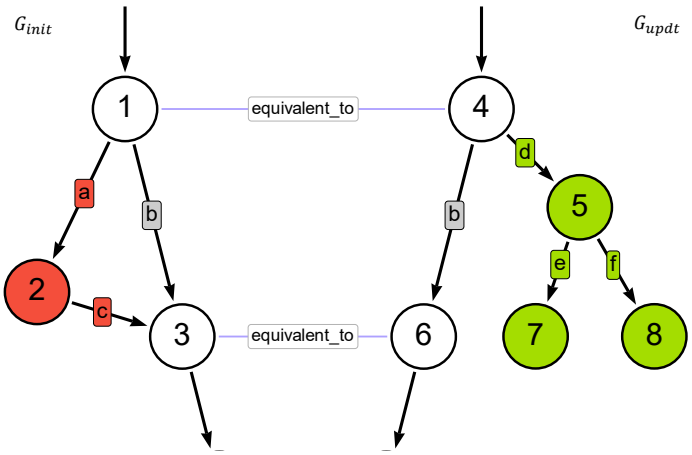


Abbildung 4.14: Ableitung topologischer Modifikationen ausgehend von dem maximalen gemeinsamen Subgraph G_{MCS}

Betrachtet werden zwei Ausschnitte der Graphen G_{init} und G_{updt}, deren gemeinsamer Subgraph G_{MCS} durch die Präsenz der $equivalent\_to$ Kanten repräsentiert wird. Exemplarisch herausgegriffen werden die Knoten 1 und 3 aus G_{init} sowie die Knoten 4 und 6 aus G_{updt}, die im vorherigen Prozessschritt als äquivalent eingestuft wurden. Zudem wurde für Knoten 2 in G_{init} kein Gegenstück in G_{updt} erkannt. Folglich ist abzuleiten, dass dieser Knoten und potenziell anschließende Subgraphen entfernt werden müssen. Gleiches gilt für den Knoten 5 in G_{updt}, aus dessen Existenz eine einfügende Modifikation abgeleitet wird.

Aus dem illustrierten Beispiel geht anschaulich hervor, dass neben dem Löschen von Knoten 2 und seiner eingehenden Kanten a besonderes Augenmerk auf die Kante c gelegt werden muss, die ebenfalls zu entfernen ist. Außerdem schließt sich im Graph G_{updt} dem Knoten 5 ein weiterer Teilgraph bestehend aus den Knoten 7 und 8 sowie den zugehörigen Kanten e und f an. Knoten 5 wird durch die Kante d mit jenem Teil des Graphs verbunden, der zu dem gemeinsamen Teilgraph G_{MCS} gehört und zu erhalten ist.

Zur Beschreibung des Sachverhalts werden das *PushOut*-Muster, das *Kontext*-Muster sowie das *Klebe*-Muster eingeführt. Diese folgen den allgemeinen Prinzipien, die in Abschnitt 3.1.1 eingeführt wurden. Das PushOut-Muster beschreibt einen Teilgraph, der aus dem bestehenden Graph gelöscht werden muss oder dem Graph hinzuzufügen

ist. Das Kontext-Muster enthält all jene Informationen, die für das korrekte Ausklinken beziehungsweise Einbetten des PushOut-Musters notwendig sind. Die verbindenden Kanten zwischen PushOut-Muster und Kontext-Muster werden schließlich im Klebe-Muster spezifiziert. Die Übersetzung in die entsprechende Notation mit zwei PushOut Operationen und passenden Erhaltungsmorphismen wird im weiteren Verlauf erläutert.

Um das PushOut-Muster zu ermitteln, werden alle Knoten gesucht, die nicht Teil des gemeinsamen Subgraphs G_{MCS} sind, aber entweder in G_{init} oder G_{updt} enthalten sind. Mengentheoretisch ausgedrückt ist nun also jene Knotenmenge von Interesse, die entweder gelöscht (damit Bestandteil der Knotenmenge von G_{init}) oder hinzugefügt (Bestandteil der Knotenmenge von G_{updt}) sind. Diese wird durch eine Differenz der Knoten aus G_{MCS} und G_{init} respektive G_{updt} ermittelt. Weiter wird der induzierte Subgraph gesucht, der sich aus den gelöschten beziehungsweise eingefügten Knoten in Bezug auf G_{init} respektive G_{updt} ergibt. Hierfür werden die zuvor eingeführten Äquivalenzkanten genutzt, um das PushOut-Muster zu ermitteln. Abb. 4.15 zeigt die gesuchten Teilgraphen, die sich für das in Abb. 4.14 eingeführte Beispiel ergeben. In rot ist das Muster dargestellt, welches aus dem Graph entfernt werden muss. Der in grün dargestellte Teilgraph muss hingegen neu eingefügt werden.

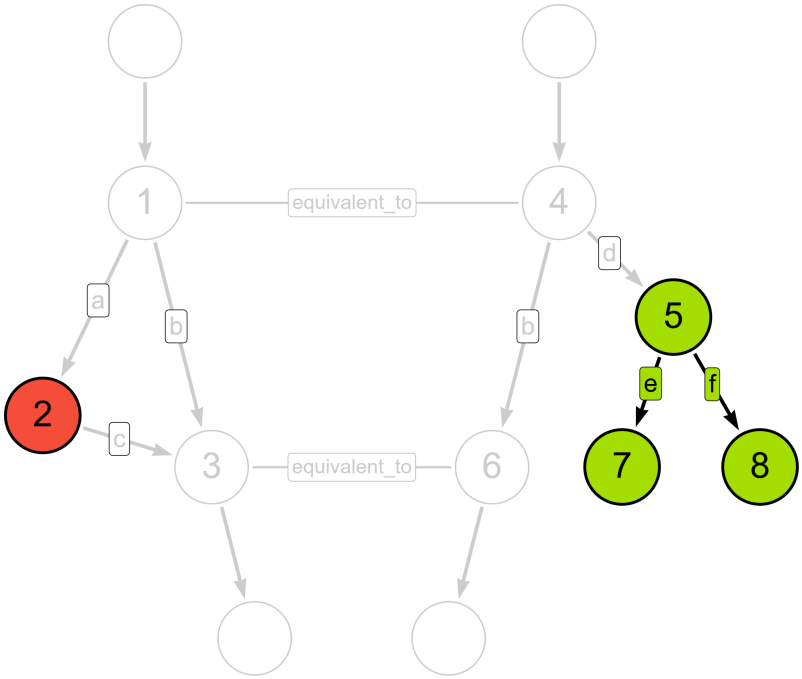


Abbildung 4.15: Ermittlung des PushOut-Musters

Die Ermittlung des PushOut-Musters wird mithilfe der in Algorithmus 4.4 gegebenen Cypher-Abfrage ausgedrückt. Das Label *TIMESTAMP* ist dabei durch den Zeitstempel des initialen Modells G_{init} zu ersetzen, um die entfernten Knoten und inzidenten Kanten zu ermitteln. In gleicher Weise wird auch der neu hinzugefügte Teilgraph ermittelt, indem das Label *TIMESTAMP* mit dem Zeitstempel des aktualisierten Graphs G_{updt} gefüllt wird.

Algorithmus 4.4: Ermittlung des PushOut-Musters

```
MATCH pa = (n:TIMESTAMP)-[:rel*0..]->(m)
WHERE NOT EXISTS ((n)-[:equivalent_to]-()) AND NOT EXISTS((m)-[:
    equivalent_to]-())
RETURN pa
```

Der $*$ Operator gibt die mögliche Pfadlänge an. Mit dem Ausdruck $-[:rel*0..]->$ wird definiert, dass die Kante des Typs rel gar nicht oder in beliebiger Anzahl zwischen den Knoten n und m existieren darf. Die Zahl möglicher Passungen wird damit deutlich größer als bei der Definition fester Pfadlängen. Daher wird nach Ausführung der Abfrage in einer nachgelagerten Prozessierung eine Normalisierung aller gefundenen Passungen des spezifizierten Musters durchgeführt, um das PushOut-Muster eindeutig, aber dennoch möglichst wenigen Knoten und Kanten zu beschreiben. Im gezeigten Beispiel resultiert die gegebene Abfrage für den Graph G_{init} im Knoten 2 sowie für den Graph G_{updt} in folgenden Passungen:

- Knoten 5
- Knoten 5, Kante e und Knoten 7
- Knoten 5, Kante f und Knoten 8
- Knoten 7
- Knoten 8

Durch Normalisierung der Knoten- und Kantenmenge aller Passungen ergibt sich schließlich das gewünschte PushOut-Muster zu Knoten 5, 7 und 8 sowie den Kanten e und f.

Anschließend wird für jeden Knoten des PushOut-Musters überprüft, ob dieser Kanten zu Knoten besitzt, die unverändert sind und demnach zum gemeinsamen Teilgraph G_{MCS} gehören. Im eingeführten Beispiel sind dies die Kanten a und d, die das zu entfernende PushOut-Muster in G_{init} an den gemeinsamen Subgraph G_{MCS} binden. Zusätzlich muss in G_{updt} die Kante d gefunden werden. Abb. 4.16 illustriert das gewünschte Ergebnis.

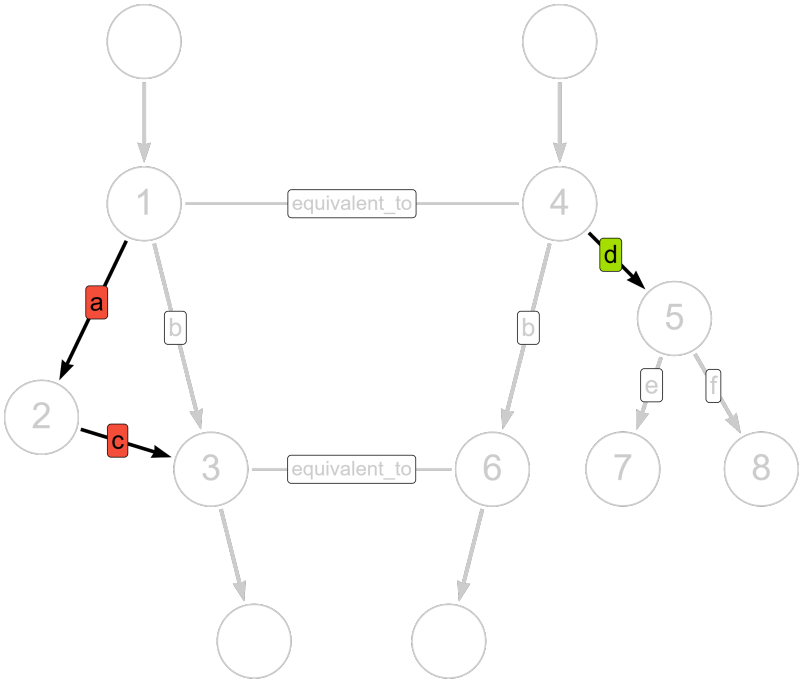


Abbildung 4.16: Ermittlung des Klebe-Musters

Die Ermittlung der Klebe-Muster wird mit den in Algorithmus 4.5 und Algorithmus 4.6 gegebenen Cypher-Abfragen umgesetzt, wobei der Knoten n einen Knoten des PushOut-Musters repräsentiert und in seiner semantischen Ausgestaltung eindeutig sein muss. Zusätzlich ist wie zuvor die korrekte Vergabe des Timestamp-Labels zu beachten, welches in den gegebenen Abfragen als Platzhalter zu verstehen ist. Da Klebe-Kanten ebenfalls gerichtet sind, müssen beide möglichen Richtungen betrachtet werden.

Algorithmus 4.5: Ermittlung der zum PushOut-Muster hin zeigenden Kanten für einen Knoten $n \in G_{PushOut}$

```
MATCH pa = (n:TIMESTAMP)<-[:rel]-(a)
WHERE EXISTS ((a)-[:equivalent_to]-())
RETURN pa
```

Algorithmus 4.6: Ermittlung der vom PushOut-Muster weg zeigenden Kanten für einen Knoten $n \in G_{PushOut}$

```
MATCH pa = (n:TIMESTAMP)-[:rel]->(a)
WHERE EXISTS ((a)-[:equivalent_to]-())
RETURN pa
```

In dem in Abb. 4.14 illustrierten Beispiel ist Kante c eine vom PushOut-Muster weg zeigende Kante, die durch die in Algorithmus 4.6 gegebene Cypher-Abfrage ermittelt wird. Außerdem stellen die Kanten a und d auf das PushOut-Muster zeigende Kanten dar, die mit Algorithmus 4.5 ermittelt werden. Letztgenannte Kanten sind auch im Rahmen des Diff-Algorthmus durch die Untersuchung der direkten Kindsknoten detektierbar.

Sofern die in Algorithmus 4.5 oder Algorithmus 4.6 gegebenen Abfragen eine erfolgreiche Passung im jeweiligen Graph G_{init} oder G_{updt} erzielen, muss abschließend ein

eindeutiges Muster ermittelt werden, mit dem der in Algorithmus 4.5 bzw. Algorithmus 4.6 als a bezeichnete Knoten eindeutig im Graph auffindbar ist. Diese Muster bilden den sogenannten Kontext, um die zu entferndenden oder einzufügenden Teilgraphen korrekt mit dem unveränderten Teil des Graphs zu verbinden. Für das in Abb. 4.14 illustrierte Beispiel sind dies die Knoten 1 und 3 für die Anbindung des zu entfernenden Teilgraphs sowie der Knoten 4 für den einzufügenden Teil. Diese Muster sind in Abb. 4.17 dargestellt.

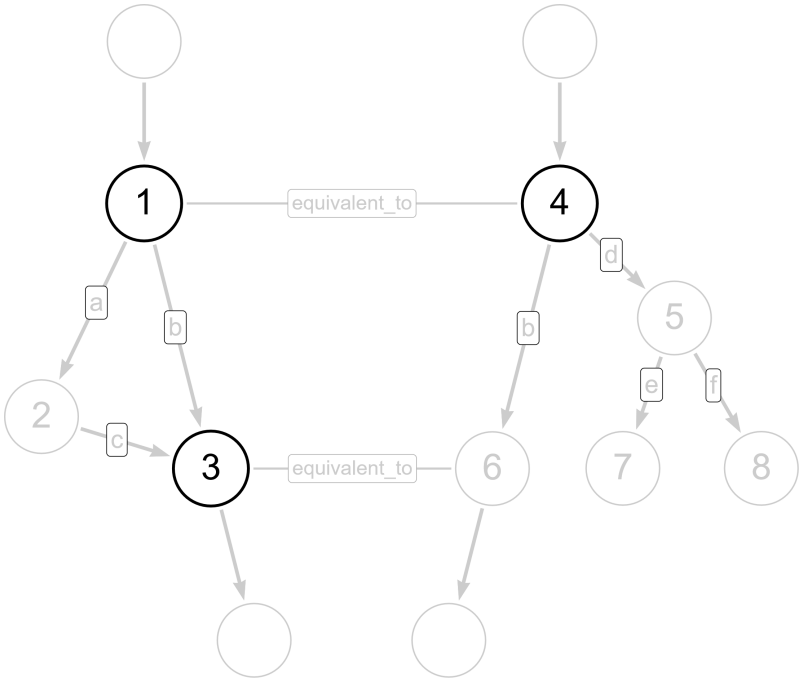


Abbildung 4.17: Ermittlung des Kontext-Musters

Im Falle von Primärknoten oder als äquivalent eingestuften Beziehungsknoten ergibt sich das zugehörige Kontext-Muster durch den Knoten selbst, sofern dieser über ein eindeutiges und über die untersuchten Versionen hinweg stabiles Merkmal besitzt. Handelt es sich bei Knoten a um einen Sekundärknoten, wird das in Abschnitt 4.6.1 erläuterte Vorgehen genutzt. Dabei wird ein eindeutiger Pfad aus dem betroffenen Sekundärknoten sowie einem Primärknoten gesucht, der Bestandteil des Graphs G_{MCS} und demnach in beiden Versionen G_{init} und G_{updt} enthalten ist.

Die Suche nach einem passenden Kontext-Muster wird mit der in Algorithmus 4.7 gegebenen Cypher-Anfrage umgesetzt. Da in vielen Fällen mehrere Primärknoten Pfade zum zu spezifizierenden Sekundärknoten a bilden können, wird die gewünschte Rückgabe auf eine Passung begrenzt.

Algorithmus 4.7: Ermittlung eines eindeutigen Kontextmusters für den Knoten a

```
MATCH pa = (a)<-[:rel*]-(p:PrimaryNode)
WHERE EXISTS ((p)-[:equivalent_to]-())
RETURN pa LIMIT 1
```

Alle Pfade zur eindeutigen Auffindung der Knoten a ergeben zusammen das Kontext-Muster. Die verbindenden Kanten zwischen Knoten des Kontext-Musters und des PushOut-Musters werden im Klebe-Muster gespeichert.

Die extrahierten Muster können nun in eine DPO-Notation überführt werden. Hierfür müssen die ermittelten PushOut-, Kontext- und Klebe-Muster entsprechend kombiniert werden. Abb. 4.18 zeigt die entsprechende Übersetzung des zuvor erläuterten Beispiels. Wichtig ist, dass eine boolesche Vereinigung über die Kontext-Muster erfolgen muss, um sämtliche Erhaltungsmorphismen richtig darzustellen. Für die Konstruktion des Mustergraphs L werden beide Kontext-Muster sowie das aus G_{init} abgeleitete PushOut- und Klebe-Muster benötigt. Der Zwischengraph I ergibt sich aus den beiden ermittelten Kontext-Mustern. Abschließend kann der Ersetzungsgraph R aus den Kontext-Mustern sowie dem PushOut- und Klebe-Muster konstruiert werden, die aus G_{updt} abgeleitet wurden.

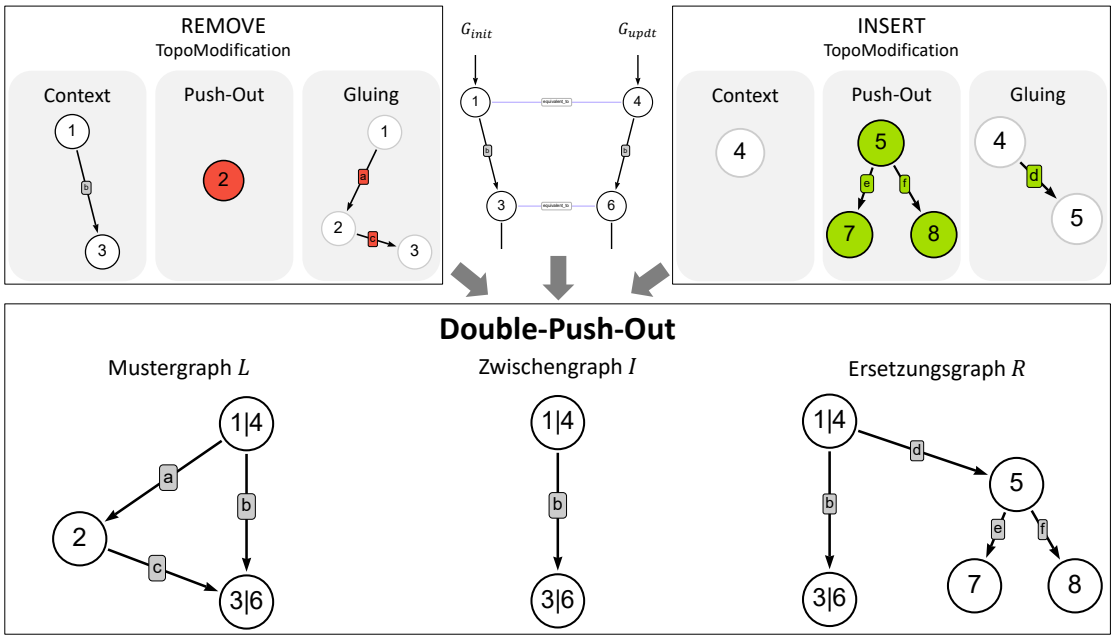


Abbildung 4.18: Übersetzung der topologischen Modifikationen in die Double Push Out Notation

Ein Beispiel für die Anwendung einer topologischen Transformationen ist in Abschnitt 4.9.2 für das Einfügen einer Modellkomponente erläutert.

Mit dem beschriebenen Verfahren wird somit eine einzige große Graphtransformation abgeleitet, die generisch für jegliche topologischen Änderungen funktioniert. Durch weitere Analysen der einzelnen Muster beziehungsweise der darin enthaltenen Knoten und Kanten können aber weitergehende Überlegungen getätigt werden, welche Bauteile von der abgeleiteten Transformation betroffen sind. Da es sich hierbei aber wiederum um eine Prozessierung handelt, die das Hinzuziehen weiterer Logik aus den jeweiligen Datenmodellen bedarf, wird diese an dieser Stelle nicht weiter beleuchtet. Hinweise und weitere Überlegungen werden allerdings später in Abschnitt 5.4 diskutiert.

4.7 Anwendung des Inkrements auf Empfängerseite

In den vorangegangenen Abschnitten wurde umfassend dargelegt, wie ein Versionsinkrement zwischen zwei Versionen eines BIM-Modells auf Grundlage der zugehörigen Graphrepräsentationen ermittelt wird und in welcher Weise die durchgeführte Modifikation formal beschrieben werden kann. Um einen umfassenden Nutzen des entwickelten Systems zu erzielen, wird nun die Anwendung des Inkrements auf eine veraltete Modellversion beziehungsweise der zugehörigen Graphrepräsentation behandelt. Das überholte Modell kann entweder lokal bei dem sendenden Client vorliegen oder im Sinne eines verteilten Systems bei einem anderen, empfangenden Nutzer vorgehalten werden. In beiden Fällen ist davon auszugehen, dass eine Modellversion vorliegt, die jener der initialen Version des Senders entspricht. Genauer wird vorausgesetzt, dass der Empfänger des Inkrements eine Graphrepräsentation G des überholten BIM-Modells vorliegen hat oder mit dem in Abschnitt 4.4 erläuterten Vorgehen erstellen kann. Dieser Graph wird im Folgenden als Arbeitsgraph bezeichnet und entspricht topologisch sowie semantisch dem Graph G_{init}, der dem Sender des Inkrements ebenfalls vorlag.

Das grundsätzliche Vorgehen zur Anwendung der topologischer Modifikationen ist in Abb. 4.19 illustriert. Wie dargelegt umfasst ein Inkrement aber zusätzlich auch noch semantische Modifikationen, die ebenfalls während der Inkrementanwendung berücksichtigt werden müssen.

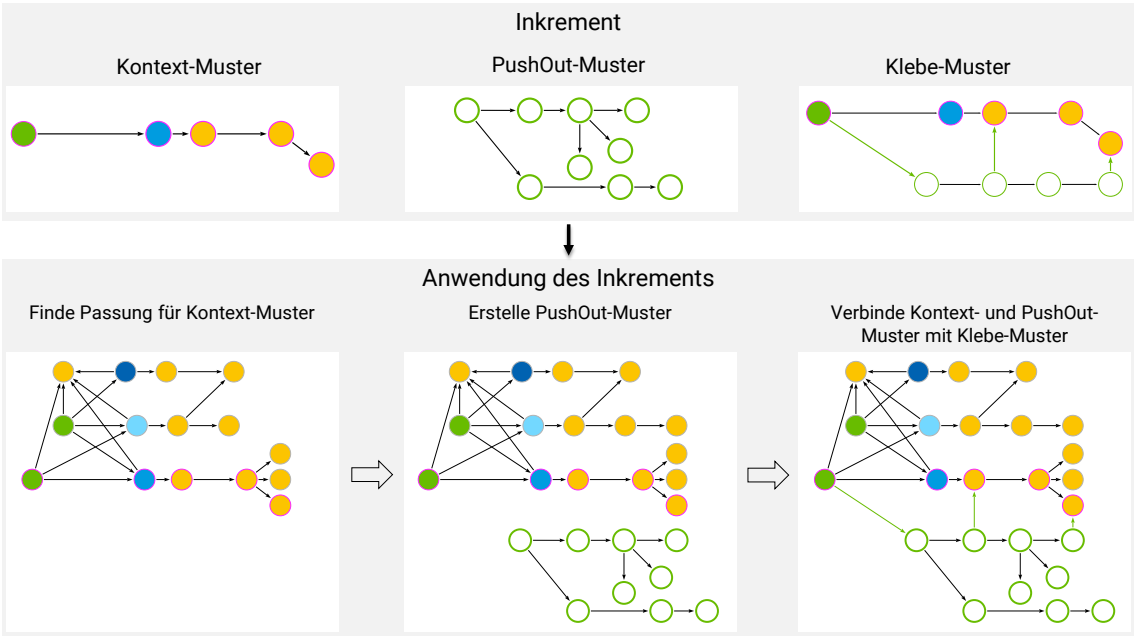


Abbildung 4.19: Bestandteile eines Versionsinkrements bei einer topologischen Modifikation

Auf empfangender Seite gestaltet sich die Anwendung des Inkrements als klassisches Problem der Graphtransformation. Das Inkrement wird dabei als Transformationsregel p bezeichnet. Notiert wird die Anwendung der Transformationsregel als $Patch(p, G)$, wobei das Inkrement p auf einen Arbeitsgraph G angewendet wird. Weiterhin beschreibt die

Notation p^{-1} die inverse Anwendung des Inkrements. Algorithmus 4.8 fasst das Vorgehen zur Anwendung des Inkrements in Pseudo-Code Notation zusammen.

Algorithmus 4.8: Verfahren zur Anwendung eines Inkrements auf eine veraltete Version

```
// wende topologische Remove Transformationen an
foreach tMod[RMV]:
    Finde Passung für PushOut-Muster + Kontext-Muster + Klebe-Muster
    Lösche alle Kanten des Klebemusters
    Lösche PushOut-Muster

// wende semantische Transformationen an
foreach sMod:
    Finde Pfad
    (Er-)Setze modifiziertes Attribut

// wende topologische Insert Transformationen an
foreach tMod[INS]:
    Finde Passung für das Kontext-Muster
    Füge PushOut-Muster ein
    Verbinde Kontext- und PushOut-Muster mit Kanten des Klebe-Musters
```

Die Anwendung des Inkrements wird nach dem All-Or-Nothing Prinzip nach dem klassischen Transaktionsmodell gemäß den in Abschnitt 2.6 erläuterten Prinzipien umgesetzt. Insbesondere das Auffinden der spezifizierten Muster im Graph ist essenziell, um die korrekte Ausführung der Modifikationen zu gewährleisten. Das Resultat nach der Anwendung des Inkrements wird als G_{res} bezeichnet. Die erfolgreiche Anwendung des Inkrements kann anschließend grundsätzlich mit den verschiedenen Betrachtungen validiert werden.

Angenommen wird, dass der resultierende Graph G_{res} mit dem aktualisierten Graph G_{updt} des Senders übereinstimmt. Folglich muss die Ermittlung des Inkrements zwischen G_{res} und G_{updt} in einem leeren Inkrement enden:

$$Diff(G_{updt}, G_{res}) \rightarrow \varnothing$$

Berechnet man das Inkrement zwischen dem initialen Graph G_{init} und dem resultierenden Graph G_{res}, so ist das gleiche Inkrement p zu erwarten:

$$Diff(G_{init}, G_{res}) = p$$

Abschließend soll die inverse Anwendung des Inkrements auf den resultierenden Graph G_{res} erneut den Graph G_{init} ergeben:

$$Patch(p^{-1}, G_{res}) = G_{init}$$

4.8 Übersetzung des Graphs in eine serialisierte Form

Wenngleich langfristig angenommen werden kann, dass dateibasierte Repräsentationen zunehmend durch direkte Kommunikationsmittel zwischen Applikationen und zentralen Speichern abgelöst werden, können für die Integration in derzeitige Softwarelösungen die Graphrepräsentationen zurück in dateibasierte Darstellungen gewandelt werden. Das Vorgehen ist den Überlegungen zur Wandlung von serialisierten Formen in Graphen sehr ähnlich. In einem ersten Schritt werden sämtliche Instanzen angelegt und damit die in den Knoten gespeicherte Information serialisiert. Anschließend werden die notwendigen Verweise, deren Informationen in den Kanten des Graphs gespeichert sind, der serialisierten Form hinzugefügt.

Die resultierende Serialisierung kann sich in der textuellen Form erheblich von der ursprünglichen Repräsentation unterscheiden, da eine beliebige Sortierung der Knoten und Kanten während der Übersetzung herangezogen wird. Ein Vergleich mit Diff- und Patch-Methoden für serialisierte Darstellungen erzielt daher keine aussagekräftigen Ergebnisse.

4.9 Beispiele für die Ermittlung eines Inkrements

Um die Funktionsweise der zuvor erläuterten Methodik zur Abstraktion von Modelländerungen im Detail zu erläutern, werden im Folgenden zwei Beispiele zur Bestimmung von Versionsinkrementen gesamtheitlich skizziert. Als Ausgangsmodell dient in allen Szenarien das Würfelbeispiel aus Abschnitt 4.4. Abschnitt 4.9.1 zeigt dabei eine rein semantische Modifikation und Abschnitt 4.9.2 das Einfügen einer weiteren Modellkomponente in ein bestehendes Modell, woraus die Notwendigkeit einer topologischen Modifikation des Graphs resultiert. Alle Modelle folgen dem IFC-Datenmodell.

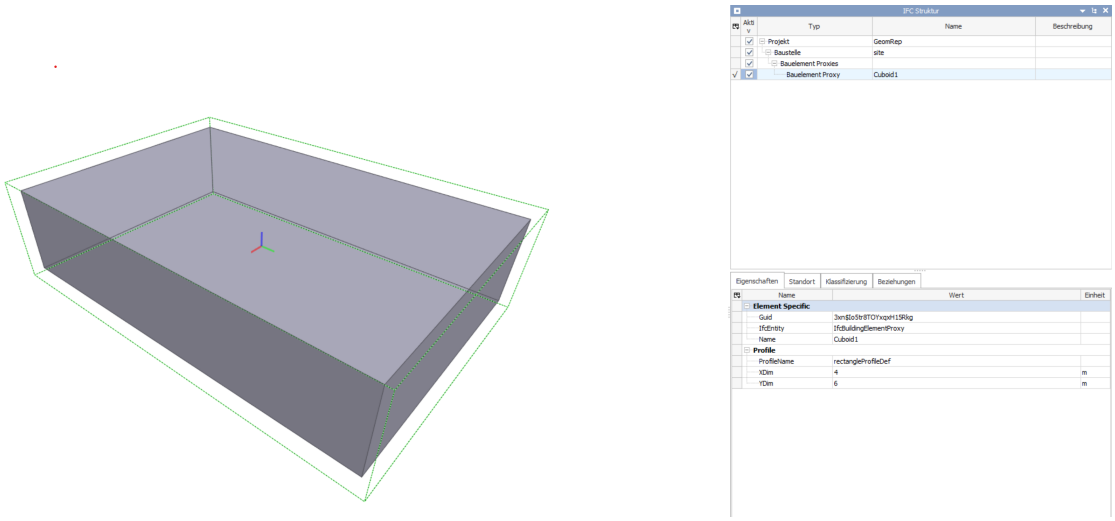


Abbildung 4.20: Visualisierung des BIM-Modells in einem Modell-Viewer (hier in DataComp BIM Vision)

Die zugehörige IFC-Datei in SPF-Serialisierung ist nochmals in Algorithmus 4.9 gegeben. Alle Knoten des entsprechenden Graphs werden mit dem Zeitstempel *ts20210119T085406* beschriftet, der das Exportdatum repräsentiert.

Algorithmus 4.9: IFC-Modell zur Veranschaulichung semantischer und topologischer Modifikationen (Wiederholung)

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [Ifc4NotAssigned]'),'2;1');
FILE_NAME('Cube_single.ifc',
  '2021-01-19T08:54:06',
  ('Sebastian Esser'),
  ('TUM'),
  'GeometryGymIFC v0.1.6.0 by Geometry Gym Pty Ltd',
  'CreateUnitTest v1.0.0.0',
  'None');

FILE_SCHEMA (('IFC4'));
ENDSEC;

DATA;
#1 = IFCCARTESIANPOINT((0.0,0.0,0.0));
#2 = IFCAXIS2PLACEMENT3D(#1,$,$);
#3 = IFCLOCALPLACEMENT($,#2);
#4 = IFCSITE('2gG1du9OH4eQ4omNtwzfn1',$,'site',$,$,#3,$,$,$,$,$,$,$,$);
#5 = IFCPROJECT('2KrDsiaI1FkRwWUa5EvoNK',$,'GeomRep',$,$,$,$,(#10),$);
#6 = IFCRELAGGREGATES('1U7z3biONu4LOUNA5m7S6j',$,$,$,#5,(#4));
#7 = IFCRECTANGLEPROFILEDEF(.AREA.,'rectangleProfileDef',$,4.0,6.0);
#8 = IFCEXTRUDEDAREASOLID(#7,$,#9,1.35);
#9 = IFCDIRECTION((0.0,0.0,1.0));
#10= IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,0.0001,#12,#13);
#11= IFCCARTESIANPOINT((0.0,0.0,0.0));
#12= IFCAXIS2PLACEMENT3D(#11,$,$);
#13= IFCDIRECTION((0.0,1.0));
#14= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',*,*,*,*,#10,$,.
    MODEL_VIEW.,$);
#15= IFCSHAPEREPRESENTATION(#14,'Body','SweptSolid',(#8));
#16= IFCPRODUCTDEFINITIONSHAPE($,$,(#15));
#17= IFCCARTESIANPOINT((2.0,5.0,1.0));
#18= IFCAXIS2PLACEMENT3D(#17,$,$);
#19= IFCLOCALPLACEMENT(#3,#18);
#20= IFCBUILDINGELEMENTPROXY('3xneIo5tr8TOYxqxH15Rkg',$,'Cuboid1',$,$,#19,
    #16,$,$);
#21= IFCRELCONTAINEDINSPATIALSTRUCTURE('3FNv6N_ur0zQ8tygS1XDuH',$,'Site','
    Site',(#20),#4);
ENDSEC;
END-ISO-10303-21;
```

Der Graph des initialen Modells G_{init} ist in Abb. 4.21 dargestellt.

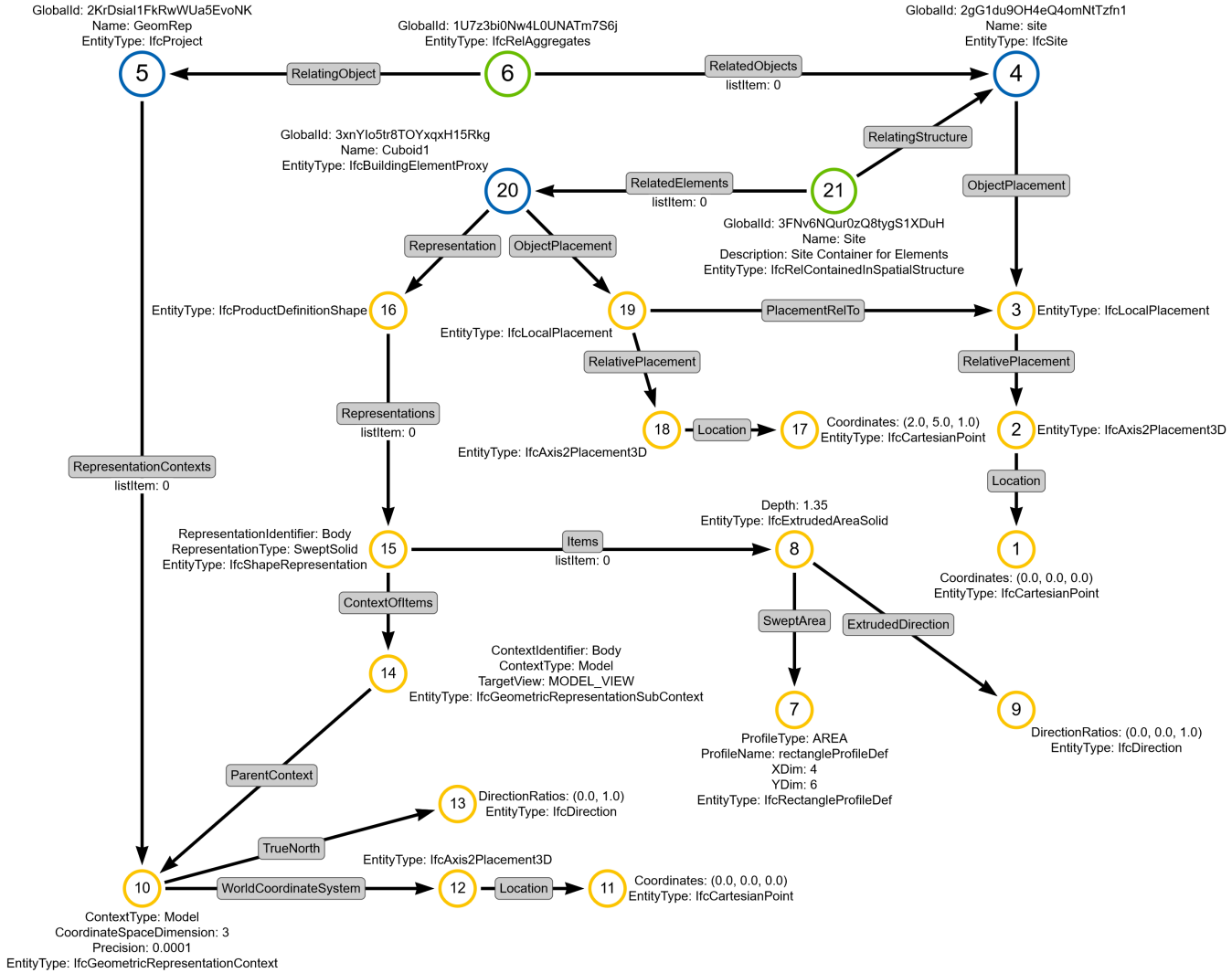


Abbildung 4.21: Visualisierung des initialen Modells, beschrieben durch den Graph G_{init}

4.9.1 Modifizieren der Bauteilhöhe von extrudierten Geometrien

Modifiziert wird die Extrusionshöhe, die in der Instanz der Enität mit der Ordnungszahl 8 gespeichert ist, welche eine Instanz der Klasse `IfcExtrudedAreaSolid` ist. Die Extrusionshöhe wird von 1.35 auf 2.50 verändert. Abb. 4.22 zeigt die visualisierten Modellversionen mit zugehörigen Maßketten sowie Algorithmus 4.10 die modifizierte Instanz in der serialisierten Form des Modells. Die übrigen Informationen der *DATA*-Sektion bleiben unverändert. Im Header hat sich aufgrund des erneuten Exports der Zeitstempel verändert.

(a) Visualisierung der initialen Modellversion

(b) Visualisierung der aktualisierten Modellversion

Abbildung 4.22: Beispiel 1: Visualisierung der Modellversionen

Algorithmus 4.10: Beispiel 1: Modifizierte Instanz im IFC-Modell

```
#8 = IFCEXTRUDEDAREASOLID(#7,$,#9, 2.50);
```

Bei dieser Änderung handelt es sich um eine reine semantische Modifikation des Knotenattributs, das die Extrusionshöhe des Quaders beschreibt. Diese wird durch eine Instanz der *SemModification* Klasse gemäß des in Abb. 4.12 eingeführten Datenmodells beschrieben. Der eindeutige Pfad zum Auffinden des zu modifizierenden Knotens ist in Abb. 4.23 gezeigt und wird in Cypher-Syntax wie in Algorithmus 4.11 formuliert ausgedrückt.

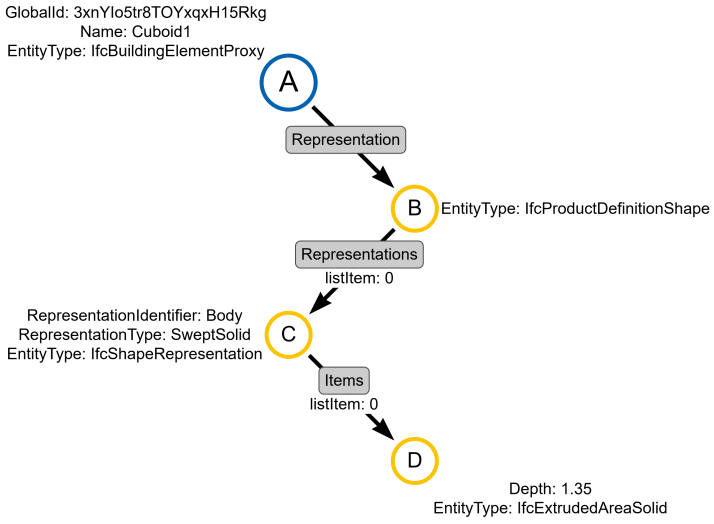


Abbildung 4.23: Beispiel 1: Darstellung des *UniquePath*-Musters zur Modifikation der Extrusionshöhe

Algorithmus 4.11: Beispiel 1: UniquePath-Muster zum Auffinden des modifizierten Knotens

```
MATCH uniquePath =
(A:PrimaryNode:ts20210119T085406{
  GlobalId: "3xnYIo5tr8TOYxqxH15Rkg",
  Name: "Cuboid1",
  EntityType: "IfcBuildingElementProxy"})
-[:rel{RelType: "Representation"}]->
(B:SecondaryNode:ts20210119T085406{
  EntityType: "IfcProductDefinitionShape"})
-[:rel{RelType:"Representations", listItem: 0}]->
(C:SecondaryNode:ts20210119T085406{
  RepresentationIdentifier: "Body",
  RepresentationType: "SweptSolid",
  EntityType: "IfcShapeRepresentation"})
-[:rel {RelType: "Items", listItem: 0}]->
(D:SecondaryNode:ts20210119T085406{
  Depth: 1.35,
  EntityType: "IfcExtrudedAreaSolid"})
```

Die Passung des Pfads ergibt sich wie folgt:

$$\varphi_N = \begin{cases} \varphi(A) \to 20 \\ \varphi(B) \to 16 \\ \varphi(C) \to 15 \\ \varphi(D) \to \ \ 8 \end{cases} \qquad \varphi_E = \begin{cases} \varphi[A \to B] \to [20 \to 16] \\ \varphi[B \to C] \to [16 \to 15] \\ \varphi[C \to D] \to [15 \to \ \ 8] \end{cases}$$

Die Modifikation des Wertes im Attribut *Depth* wird mit dem *SET*-Befehl realisiert. Um den entsprechenden Knoten korrekt anzusprechen, wurden in der in Algorithmus 4.11 gegebenen Musterdefinition die Variablen A, B, C und D für die Knoten des Pfades vergeben. D bezeichnet jenen Knoten, der das zu modifizierende Attribut trägt. Der initiale Attributwert wird ebenfalls im Muster spezifiziert.

Algorithmus 4.12: Beispiel 1: Modifikation des Knotenattributs

```
MATCH uniquePath // wie zuvor beschrieben
SET D.Depth = 2.50
```

Abschließend werden alle Knoten des Graphs auf den neuen Zeitstempel aktualisiert.

4.9.2 Einfügen einer neuen Modellkomponente

Im zweiten Beispiel wird das Einfügen einer weiteren Modellkomponente betrachtet. Beide Modellversionen sind in Abb. 4.24 visualisiert. Hinzugefügt wurde in der aktualisierten Version eine zweite Modellkomponente, die im Folgenden als *Cylinder1* bezeichnet wird und geometrisch durch einen Zylinder mit Radius 2.0 und einer Höhe von 4.1 modelliert wird. Die neue Modellkomponente ist in grün hervorgehoben und das unveränderte Bauteil in blau dargestellt. Die zugehörigen Graphen werden erneut als G_{init} und G_{updt} bezeichnet und mit den Zeitstempeln $ts20210119T085406$ bzw. $ts20210623T090002$ in der Graphdatenbank identifiziert.

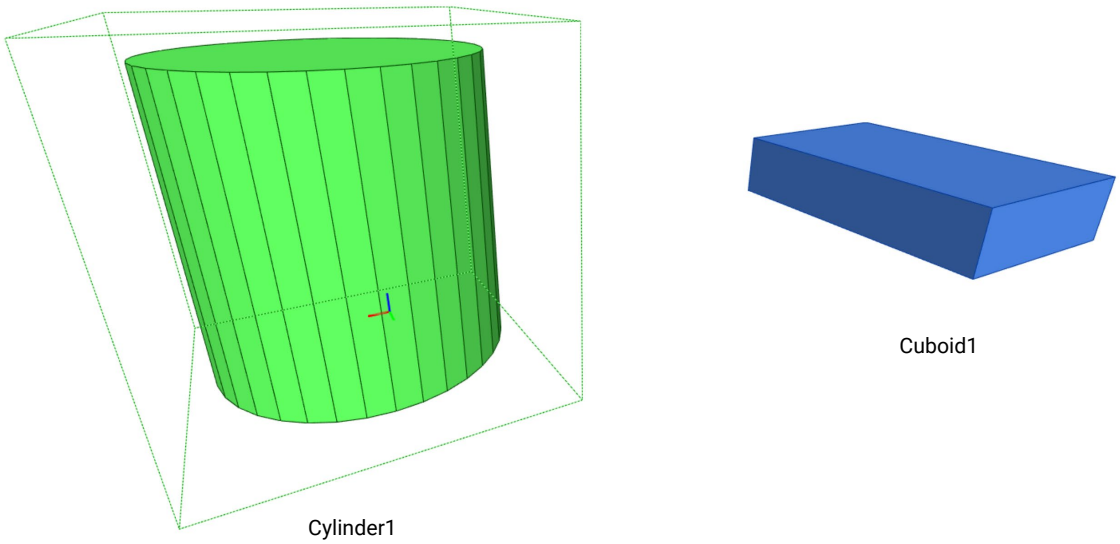


Abbildung 4.24: Beispiel 2: Visualisierung des modifizierten Modells, beschrieben durch den Graph G_{updt}

Algorithmus 4.13 zeigt die serialisierte Form des modifizierten IFC-Modells, Abb. 4.25 den zugehörigen Graphen. Hinzugekommen sind die Instanzen 22 bis 29. Zudem wurde die ausgehenden Beziehungen von Instanz 21 um einen Verweis auf Instanz 29 erweitert.

An dieser Stelle sei erwähnt, dass für die gegebenen serialisierten Formen auch eine pure Textversionierung die Änderung detektieren könnte. Diese Tatsache wird zu Zwecken der Anschaulichkeit gewählt. Die in Abschnitt 2.8 beschriebenen Abhängigkeiten von der Serialisierungsreihenfolge gelten aber weiterhin.

Algorithmus 4.13: Beispiel 2: Modifiziertes IFC-Modell mit weiterer Modellkomponente

```
ISO-10303-21;
HEADER;
// header wie zuvor, aber mit neuem Zeitstempel: 2021-06-23T09:00:02

ENDSEC;

DATA;
#1 = IFCCARTESIANPOINT((0.0,0.0,0.0));
#2 = IFCAXIS2PLACEMENT3D(#1,$,$);
#3 = IFCLOCALPLACEMENT($,#2);
#4 = IFCSITE('2gG1du9OH4eQ4omNtwzfn1',$,'site',$,$,#3,$,$,$,$,$,$,$,$);
#5 = IFCPROJECT('2KrDsiaI1FkRwWUa5EvoNK',$,'GeomRep',$,$,$,$,(#10),$);
#6 = IFCRELAGGREGATES('1U7z3biONu4LOUNA5m7S6j',$,$,$,#5,(#4));
#7 = IFCRECTANGLEPROFILEDEF(.AREA.,'rectangleProfileDef',$,4.0,6.0);
#8 = IFCEXTRUDEDAREASOLID(#7,$,#9,1.35);
#9 = IFCDIRECTION((0.0,0.0,1.0));
#10= IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,0.0001,#12,#13);
#11= IFCCARTESIANPOINT((0.0,0.0,0.0));
#12= IFCAXIS2PLACEMENT3D(#11,$,$);
#13= IFCDIRECTION((0.0,1.0));
#14= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',*,*,*,*,#10,$,.
    MODEL_VIEW.,$);
#15= IFCSHAPEREPRESENTATION(#14,'Body','SweptSolid',(#8));
#16= IFCPRODUCTDEFINITIONSHAPE($,$,(#15));
#17= IFCCARTESIANPOINT((2.0,5.0,1.0));
#18= IFCAXIS2PLACEMENT3D(#17,$,$);
#19= IFCLOCALPLACEMENT(#3,#18);
#20= IFCBUILDINGELEMENTPROXY('3xneIo5tr8TOYxqxH15Rkg',$,'Cuboid1',$,$,#19,
    #16,$,$);
#21= IFCRELCONTAINEDINSPATIALSTRUCTURE('3FNv6N_ur0zQ8tygS1XDuH',$,'Site','
    Site',(#20, #29),#4);

#22= IFCCIRCLEPROFILEDEF(.AREA.,'CircleProfileDef',$,2.0);
#23= IFCEXTRUDEDAREASOLID(#22,$,#9,4.10);
#24= IFCSHAPEREPRESENTATION(#14,'Body','SweptSolid',(#23));
#25= IFCPRODUCTDEFINITIONSHAPE($,$,(#24));
#26= IFCCARTESIANPOINT((14.0,5.0,1.0));
#27= IFCAXIS2PLACEMENT3D(#26,$,$);
#28= IFCLOCALPLACEMENT(#3,#27);
#29= IFCBUILDINGELEMENTPROXY('3uuPRBnzbDKvY53sXt1d9w',$,'Cylinder1',$,$,#28
    ,#25,$,$);
ENDSEC;
END-ISO-10303-21;
```

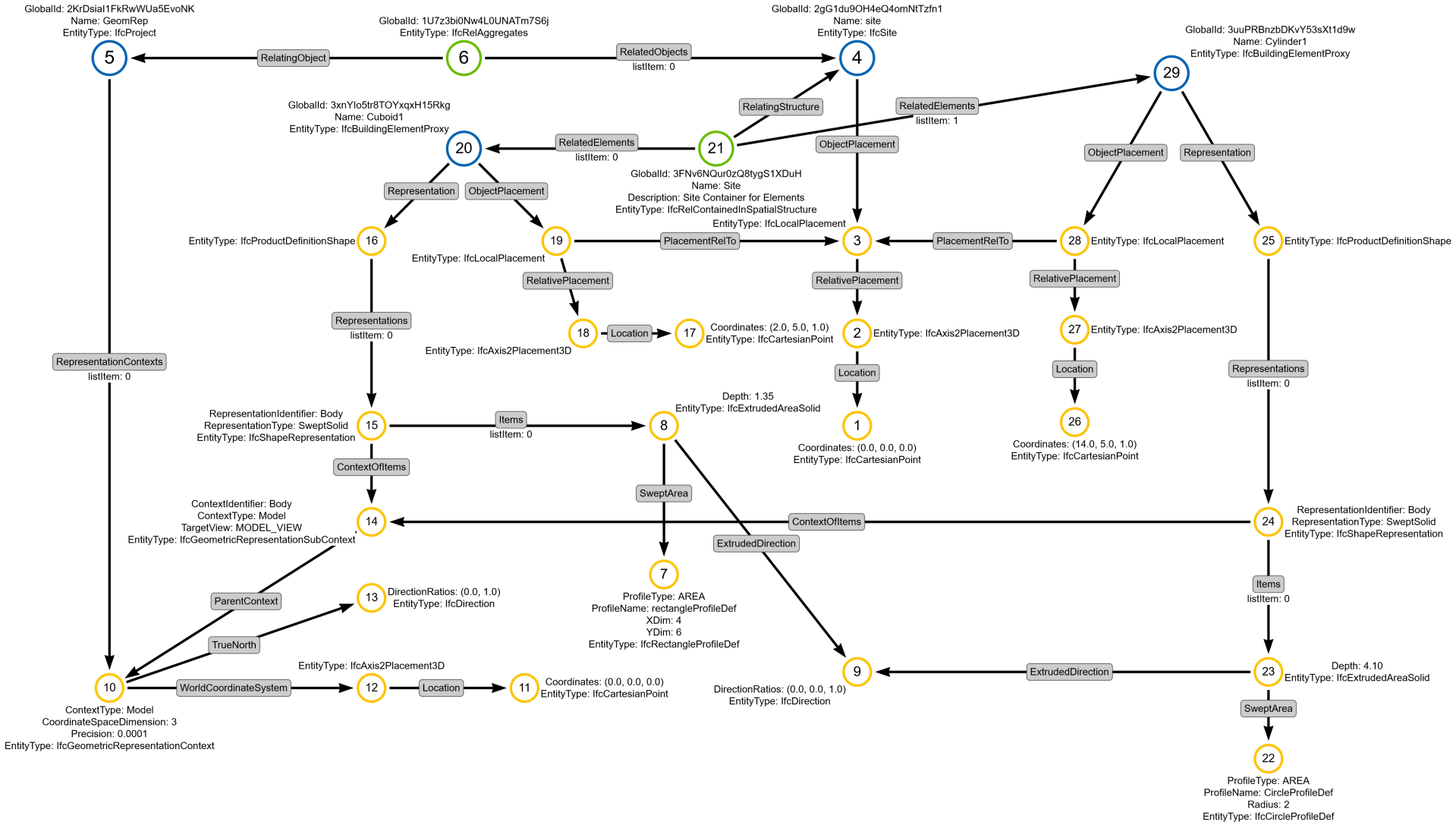


Abbildung 4.25: Beispiel 2: Graph G_{updt} des aktualisierten Modells

Im Gegensatz zum in Abschnitt 4.9.1 dargestellten Problem geht die nun behandelte Änderung mit topologischen Modifikationen des Graphs einher, die mit der Klasse *Topo-Modification* gemäß des in Abb. 4.12 gegebenen Datenmodells zur Beschreibung eines Versionsinkrements erfasst werden. Abb. 4.26 illustriert das Kontext-Muster, Abb. 4.27 das PushOut-Muster sowie Abb. 4.28 das notwendige Klebe-Muster.

Im ersten Schritt wird eine Passung des Kontextmusters im bestehenden Graph G_{init} gesucht. Die Knoten A, G und H verfügen über das eindeutige Merkmal *GlobalId*, womit eine eindeutige Passung im Graph erzielt wird. Die übrigen Sekundärknoten werden durch die Angabe der Pfade zu einem passenden Primärknoten ebenfalls eindeutig adressiert.

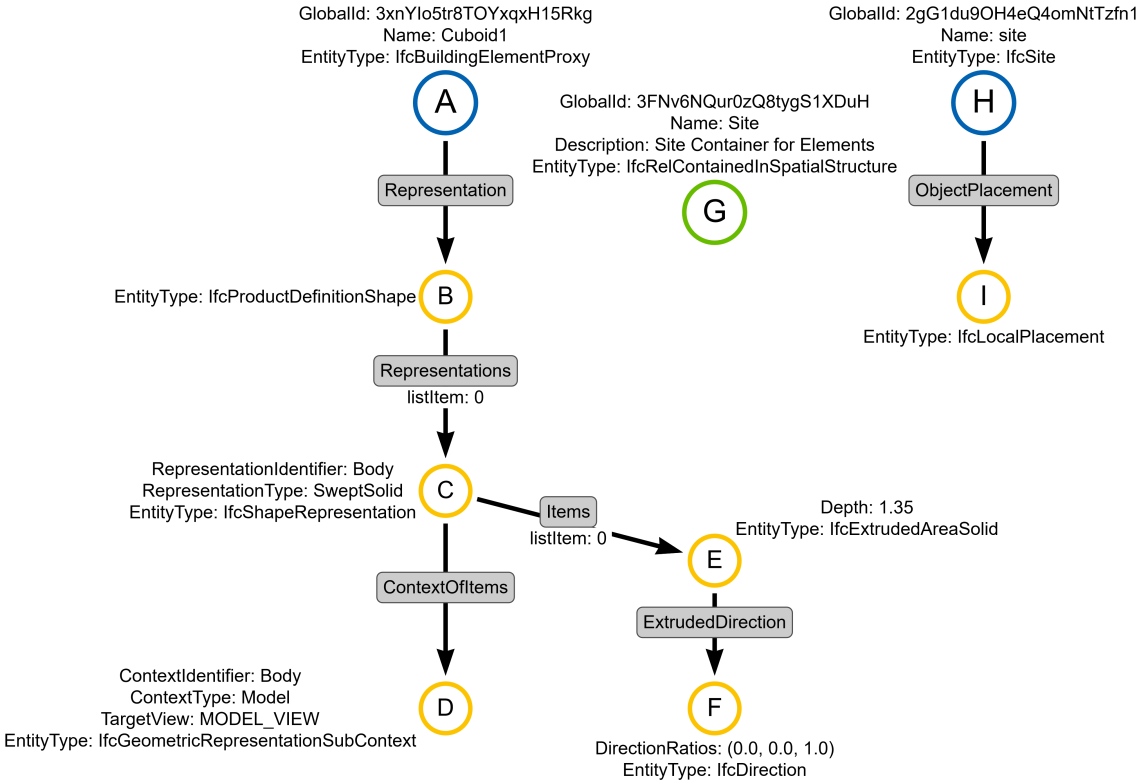


Abbildung 4.26: Beispiel 2: Kontext-Muster

Die Passung des spezifizierten Kontext-Musters auf den in Abb. 4.25 illustrierten Graph ergibt sich erwartungsgemäß wie folgt:

$$\varphi_N = \begin{cases} \varphi(A) \to 20 \\ \varphi(B) \to 16 \\ \varphi(C) \to 15 \\ \varphi(D) \to 14 \\ \varphi(E) \to 8 \\ \varphi(F) \to 9 \\ \varphi(G) \to 21 \\ \varphi(H) \to 4 \\ \varphi(I) \to 3 \end{cases} \qquad \varphi_E = \begin{cases} \varphi[A \to B] \to [20 \to 16] \\ \varphi[B \to C] \to [16 \to 15] \\ \varphi[C \to D] \to [15 \to 14] \\ \varphi[C \to E] \to [15 \to 8] \\ \varphi[E \to F] \to [8 \to 9] \\ \varphi[H \to I] \to [4 \to 3] \end{cases}$$

Es existiert demnach eine bijektive Abbildung zwischen den Knoten- und Kantenmengen. Damit spezifiziert das Kontext-Muster wie gewünscht einen homomorphen Teilgraph zu G_{init}. Liegt eine erfolgreiche Passung vor, wird wie in Abschnitt 4.7 beschrieben der im PushOut-Muster spezifizierte Teilgraph hinzugefügt. Die Benennung der Knoten kann dabei frei gewählt werden, beeinflusst aber später die Ausprägung des Kontext-Musters, welches den neu hinzugefügten Teilgraph mit dem bestehenden Graph (oder genauer mit der Passung des Kontext-Musters verknüpft).

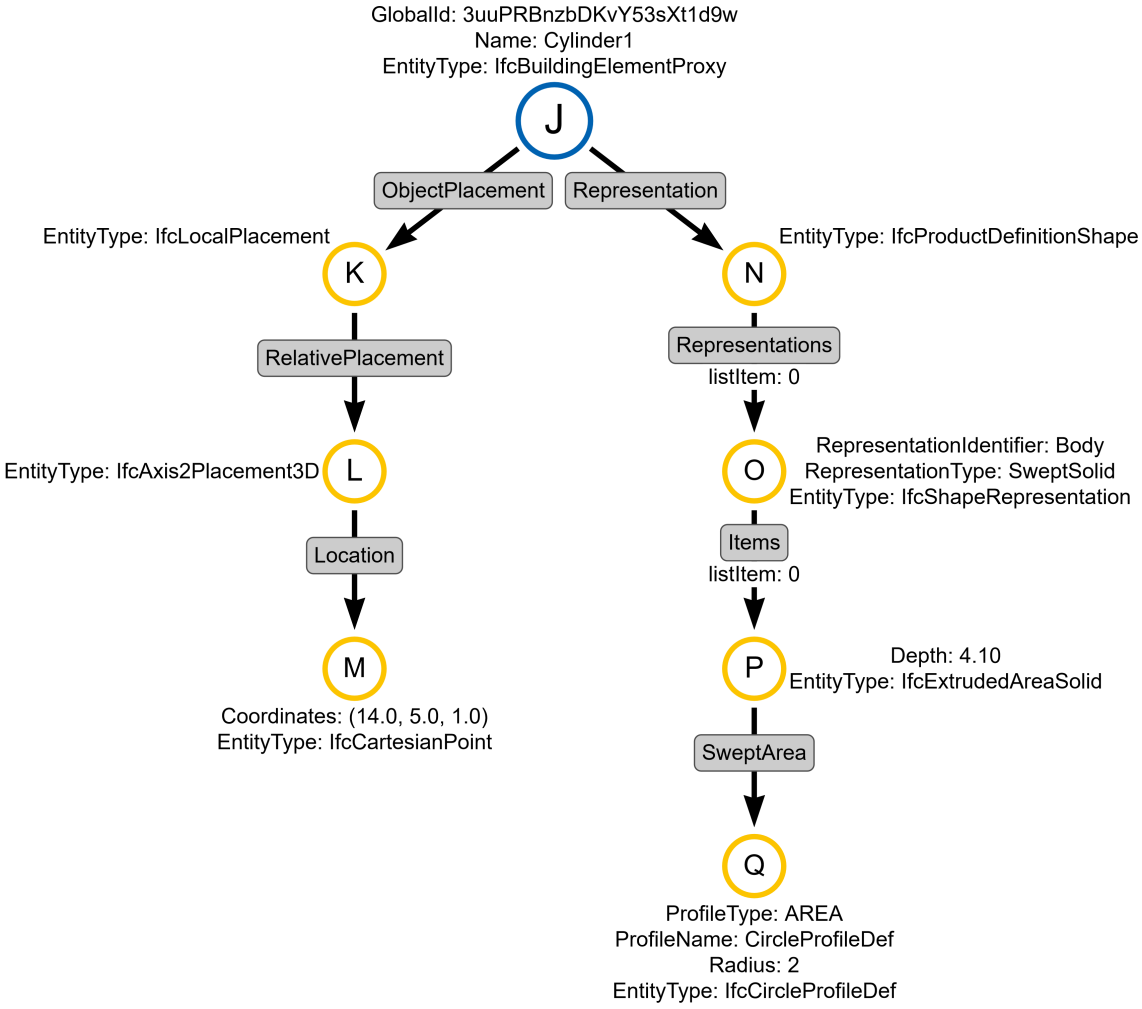


Abbildung 4.27: Beispiel 2: PushOut-Muster

Abschließend wird durch das Einfügen der im Klebe-Muster definierten Kanten das PushOut-Muster mit dem bestehenden Graph verbunden. In diesem Muster wird auf die erneute Spezifikation aller Knotenattribute verzichtet, da diese bereits in den vorangegangenen Schritten zur Passung des Kontext-Musters sowie mit dem Einfügen des PushOut-Musters spezifiziert wurden. Sofern die einzelnen Schritte zur Anwendung des Inkrements in mehrere Teilschritte aufgeteilt werden soll, müssen erneut alle Attribute derart im Muster angegeben werden, dass eine eindeutige Passung erfolgen kann.

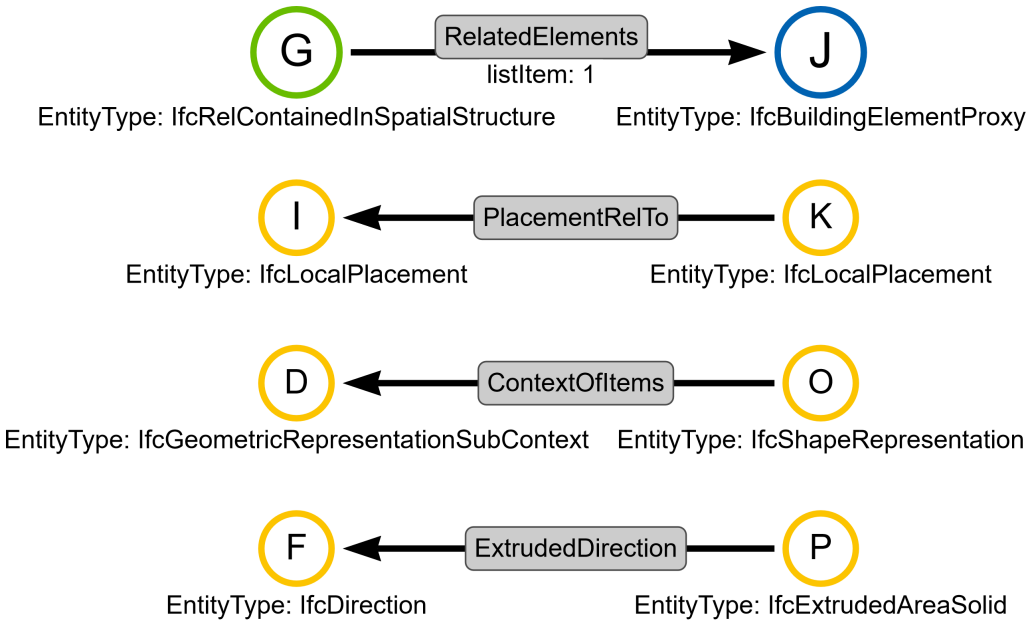


Abbildung 4.28: Beispiel 2: Klebe-Muster

Abschließend muss erneut das *timestamp*-Label aller Knoten, die das aktualisierte Modell repräsentieren, angepasst werden. Dies erfolgt analog zum in Abschnitt 4.9.1 gezeigten Vorgehen.

4.10 Globale topologische Modifikationen

Der in Abschnitt 4.5 beschriebene Ansatz zur Ermittlung des gemeinsamen Subgraphs zwischen zwei Graphen G_{init} und G_{updt} untersucht wie beschrieben in jedem Schritt stets die direkten Kindknoten.

Besonderes Augenmerk soll im Folgenden auf Situationen gerichtet werden, bei denen lediglich Änderungen in den Kantenmengen der untersuchten Graphen vorgenommen wurden. Um die Problemstellung zu motivieren, werden beispielhaft zwei Graphen G_{init} und G_{udpt} betrachtet, die in Abb. 4.29 und Abb. 4.30 gegeben sind. Diese zeigen einen Auszug eines größeren Modells und beinhalten Informationen zur Positionierung von Objekten mithilfe kartesischer Koordinatensysteme. Die Positionen im zweidimensionalen Raum werden in Abhängigkeit zu einem Referenzsystem ausgedrückt. Das hierarchisch höchste Koordinatensystem bildet das für den Modellraum global gültige System.

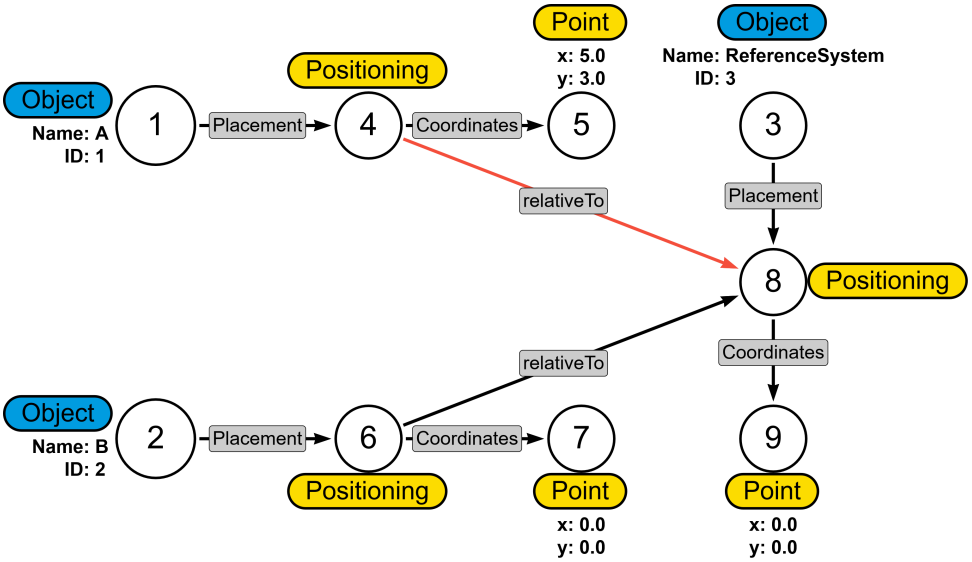


Abbildung 4.29: Initialer Graph G_{init} des fiktiven Positionierungsbeispiels

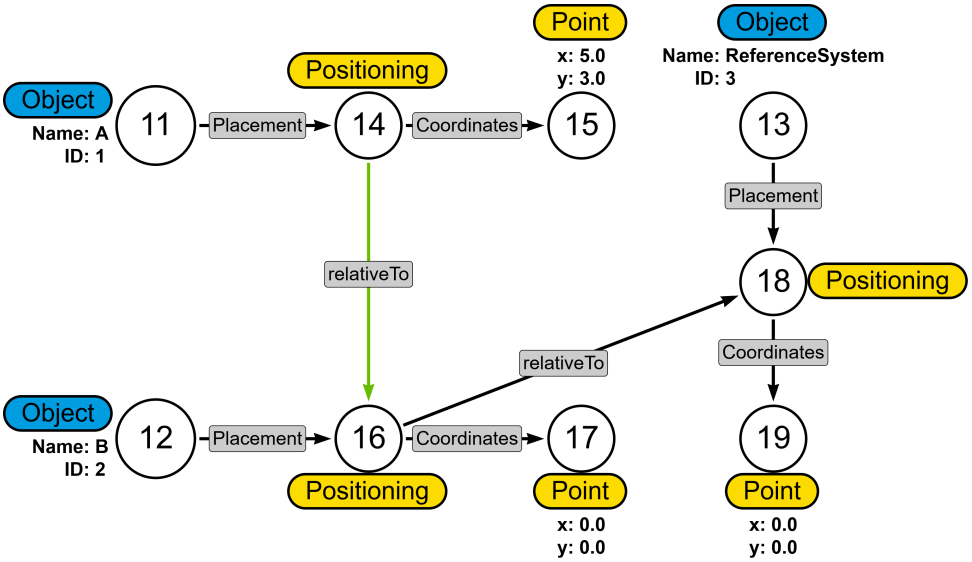


Abbildung 4.30: Modifizierter Graph G_{updt} des fiktiven Positionierungsbeispiels

Die Graphen G_{init} und G_{updt} repräsentieren jeweils drei Objekte *A*, *B* sowie *Reference-System*, die als Primärknoten mit eindeutiger ID modelliert sind. Für die Erklärungen sind die Knoten in G_{init} mit den Ziffern $1-9$ sowie die Knoten in G_{updt} mit $11-19$ beschriftet. Beide Graphen liefern in der vorliegenden Form die gleiche Interpretation. Offensichtlich ist allerdings, dass es einen Unterschied in den Kanten beider Graphen gibt. Besonderer Fokus gilt daher nun jenen Kanten, die eine relative Positionierung repräsentieren. Dies sind im Graph G_{init} die Kanten $4 \rightarrow 8$ sowie $6 \rightarrow 8$ und im Graph G_{updt} die Kanten $14 \rightarrow 16$ sowie $16 \rightarrow 18$.

Weitergehend betrachtet wird nun das Resultat, welches eine semantischen Modifikation des Attributwerts *x* an den Knoten $7 \in G_{init}$ beziehungsweise $17 \in G_{updt}$ von 0.0 auf 1.0 nach sich ziehen würde. Wird diese semantische Änderung in G_{init} vorgenommen, führt die Interpretation des Graphs für Objekt *A* weiterhin zu einer Positionierung in $(5.0, 3.0)$,

wohingegen die Lage von Objekt *B* zu $(1.0, 0.0)$ ergeben würde. Betrachtet man die gleiche semantische Modifikation nun im Graph G_{updt}, so wird die Lage des Objekts *B* zu $(1.0, 0.0)$ evaluiert. Da Objekt *A* im Graph G_{updt} allerdings das Objekt *B* als Referenz für seine Positionierung verwendet, wird die Platzierung von Objekt *B* zu $(6.0, 3.0)$ berechnet.

Für das skizzierte Beispiel wird das in Abschnitt 4.5 letztgenannte Kriterium relevant, unter denen zwei Knoten und die zugehörigen Kanten des aktuellen Traversierungsschritts als äquivalent eingestuft und damit in den gemeinsamen Subgraph G_{MCS} aufgenommen werden. Im allgemeinen Fall wurde für die Reihenfolge der Tiefentraversierung festgelegt, dass diese mit einem Primärknoten beginnt, um das erste Knotenpaar aufgrund eindeutigen, persistenten Merkmale zu ermitteln. Die Reihenfolge, in der einzelne Primärknoten und die daran anschließenden Sekundärknoten durchlaufen werden, kann aber einen Effekt auf die resultierende Transformationsregel haben, was im Folgenden diskutiert werden soll. Für das gegebene Beispiel werden alle drei möglichen Einstiegspunkte betrachtet und die Traversierungsreihenfolge tabellarisch notiert.

4.10.1 Start der Traversierung bei Knoten mit ID 1

Tabelle 4.1 stellt die Traversierungsreihenfolge dar, wenn die Knoten 1 und 9 aufgrund ihrer übereinstimmenden ID als Einstiegspunkt für die tiefenbasierte Traversierung gewählt werden.

n_{init}	n_{updt}	Bemerkung
1	11	
4	14	
5	15	
8	16	
9	17	
2	12	Ausgehende Kante auf Knoten 6 und 16 mit gleichem *relType*. Knoten 16 wurde aber bereits zuvor in ein anderes Äquivalenzpaar hinzugefügt. $\Rightarrow$ Kante wird nicht in G_{MCS} aufgenommen.
3	13	Ausgehende Kante auf Knoten 8 und 18 mit gleichem *relType*. Knoten 8 wurde aber bereits zuvor in anderes Äquivalenzpaar hinzugefügt. $\Rightarrow$ Kante wird nicht in G_{MCS} aufgenommen.

Tabelle 4.1: Traversierungsreihenfolge bei Start von Knoten mit ID 1

Aus diesen Knotenpaaren und traversierten Kanten ergibt sich der gemeinsame Subgraph $G_{MCS,1}$ zu dem in Abb. 4.31 dargestellten Graph. Die Nummerierungen der Knoten $i|j$ zeigt die jeweilige Passung des Musters in den Graphen G_{init} und G_{updt} an, wobei i die Knotennummer im initialen Graph und j die Knotennummer im aktualisierten Graph spezifiziert.

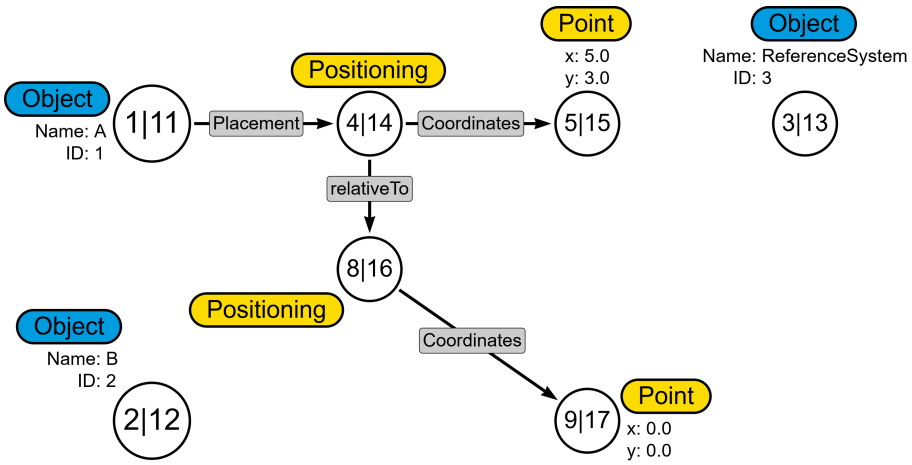
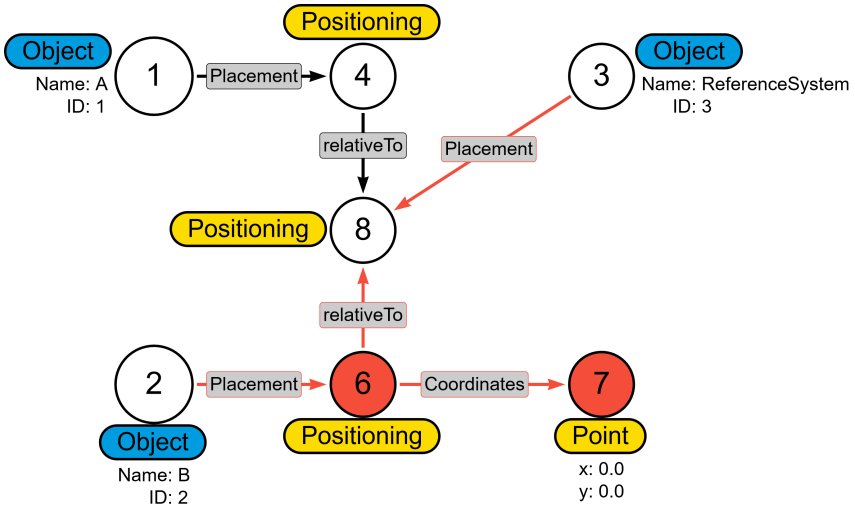
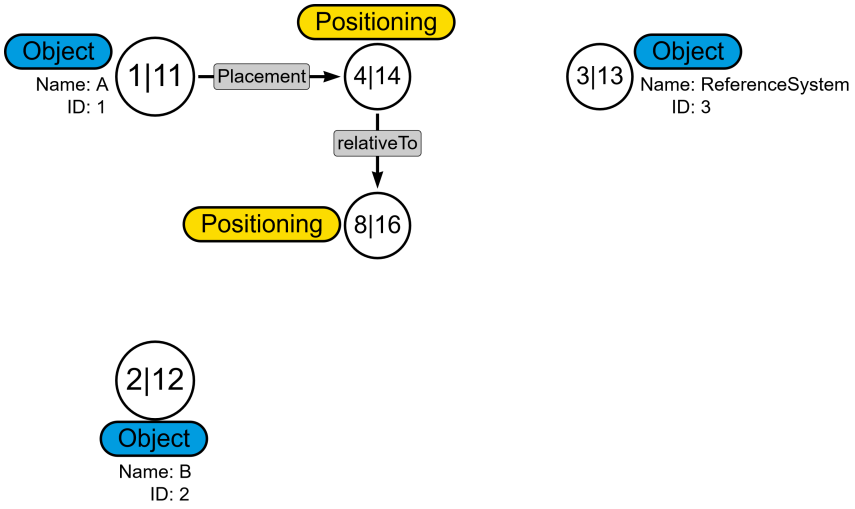


Abbildung 4.31: Gemeinsamer Subgraph G_{MCS} für Traversierungsstart in Knoten mit ID 1

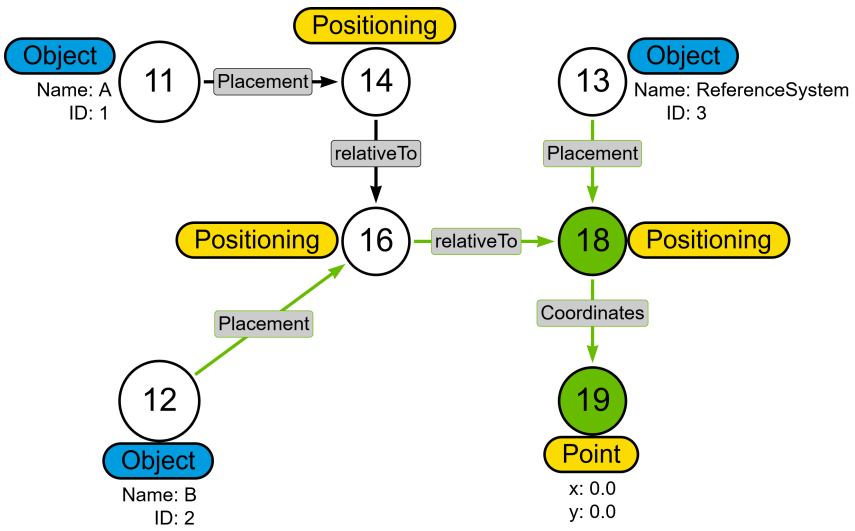
Daraus folgt die Formulierung einer topologischen Transformation gemäß des in Abschnitt 4.6.2 beschriebenen Verfahrens. Die Knoten 6 und 7 sowie die Kanten $2 \rightarrow 6$, $6 \rightarrow 8$ und $3 \rightarrow 8$ müssen entfernt werden. Zusätzlich müssen die Knoten 18 und 19 sowie die Kanten $12 \rightarrow 16$, $13 \rightarrow 18$ und $16 \rightarrow 18$ neu hinzugefügt werden. Daraus folgt die in Abb. 4.32 dargestellte Transformation in DPO-Notation.



(a) Left-Hand-Side



(b) Interface



(c) Right-Hand-Side

Abbildung 4.32: Resultierende Graphtransformation für Traversierungsstart in Knoten mit ID 1

4.10.2 Start der Traversierung bei Knoten mit ID 2

Im zweiten Fall wird untersucht, welcher gemeinsame Subgraph ermittelt wird, wenn der Primärknoten mit ID 2 als Start der rekursiven Tiefentraversierung genutzt wird. Tabelle 4.2 stellt die Traversierungsreihenfolge dar, wenn die beiden Knoten 2 und 12 aufgrund ihrer übereinstimmenden ID als Einstiegspunkt der Rekursion gewählt werden.

n_{init}	n_{updt}	Bemerkung
2	12	
6	16	
7	17	
8	18	
9	19	
1	11	
4	14	Ausgehende Kante auf Knoten 6 und 16 mit gleichem *relType*. Knoten 16 wurde aber bereits zuvor in ein anderes Äquivalenzpaar hinzugefügt. $\Rightarrow$ Kante wird nicht in G_{MCS} aufgenommen.
5	15	
3	13	Ausgehende Kante auf Knoten 8 und 18 mit gleichem *relType*. Dieses Knotenpaar wurde zuvor bereits als äquivalent eingestuft $\Rightarrow$ Kante wird in G_{MCS} aufgenommen.

Tabelle 4.2: Traversierungsreihenfolge bei Start von Knoten mit ID 2

Bei dieser Traversierung wurden alle Knoten erfolgreich besucht. Die Kanten $4 \rightarrow 6$ sowie $14 \rightarrow 16$ wurden allerdings nicht in den gemeinsamen Subgraph G_{MCS} aufgenommen, da Knoten 16 zuvor bereits mit Knoten 6 in Relation gesetzt wurde. Der ermittelte gemeinsame Subgraph G_{MCS} ergibt sich demnach zu dem in Abb. 4.33 dargestellten Graph.

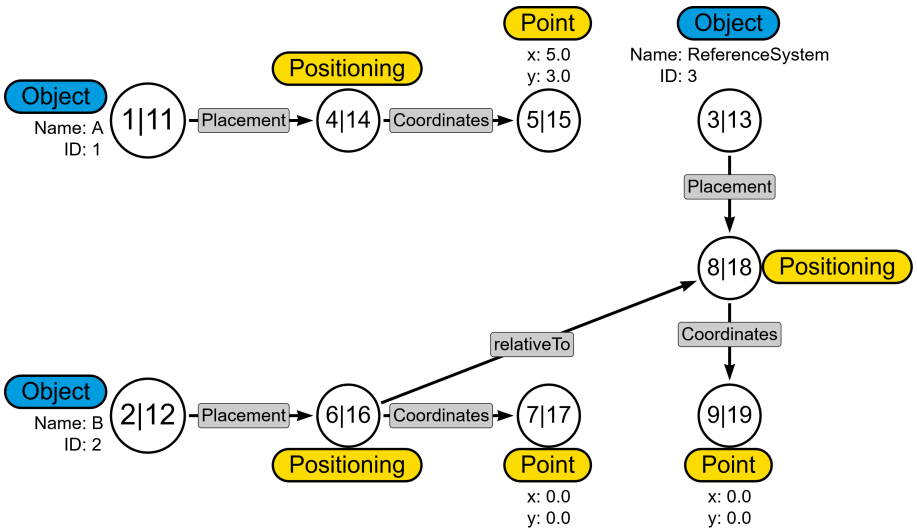
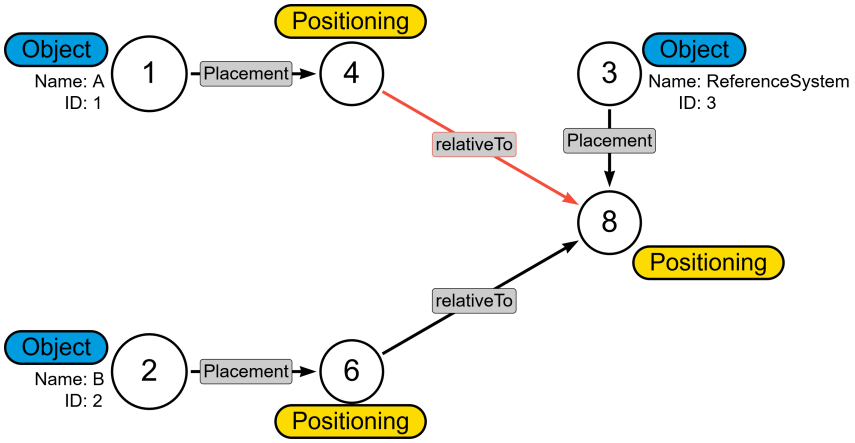
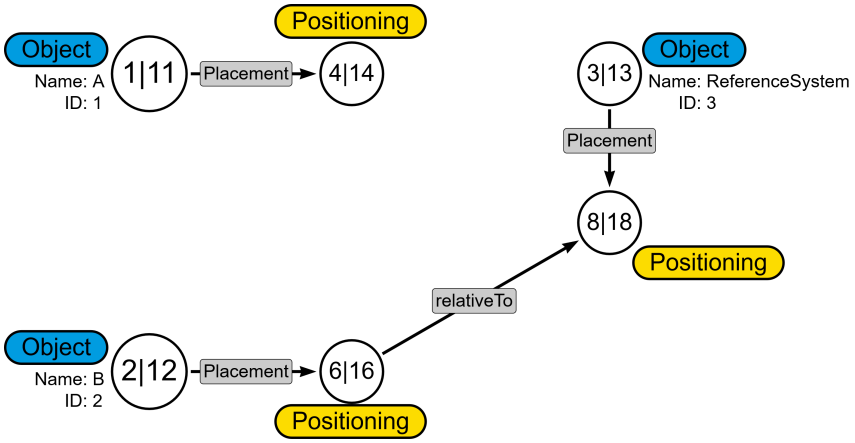


Abbildung 4.33: Gemeinsamer Subgraph G_{MCS} für Traversierungsstart in Knoten mit ID 2

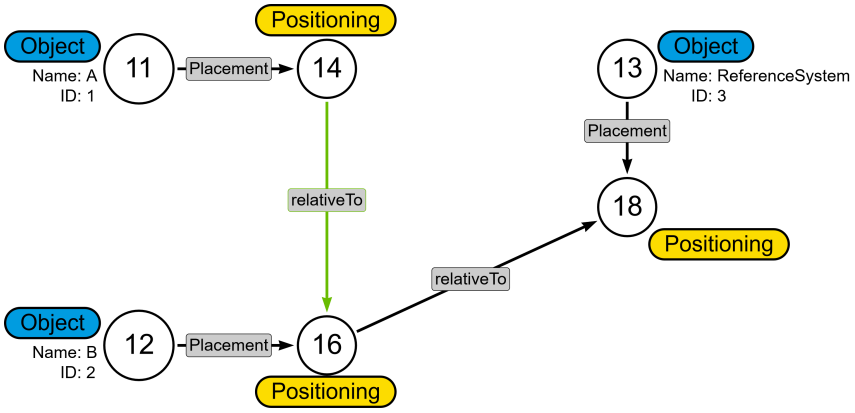
Die daraus abgeleitete Graphtransformation muss demnach lediglich die Kante $3 \rightarrow 8$ entfernen und eine neue Kante $14 \rightarrow 16$ einfügen. Da es sich bei den Knoten 4, 8 und 16 um Sekundärknoten handelt, müssen die Muster der Transformationsregel so formuliert werden, dass deren Passungen eindeutig sind. Dies wird wie zuvor beschrieben durch das Hinzufügen von Pfaden zu Primärknoten erreicht. Die resultierende Transformation ist in DPO-Notation in Abb. 4.34 dargestellt.



(a) Left-Hand-Side



(b) Interface



(c) Right-Hand-Side

Abbildung 4.34: Resultierende Graphtransformation für Traversierungsstart in Knoten mit ID 2

4.10.3 Start der Traversierung bei Knoten mit ID 3

Im dritten Fall wird die Traversierung in den Knoten 3 und 13 gestartet. Tabelle 4.3 stellt die Traversierungsreihenfolge dar, wenn die beiden Knoten 3 und 13 aufgrund ihrer übereinstimmenden ID als Einstiegspunkt der Rekursion gewählt werden.

n_{init}	n_{updt}	Bemerkung
3	13	
8	18	
9	19	
1	11	
4	14	Ausgehende Kante auf Knoten 8 und 16 mit gleichem *relType*. Knoten 8 wurde aber bereits zuvor in ein anderes Äquivalenzpaar hinzugefügt. $\Rightarrow$ Kante wird nicht in G_{MCS} aufgenommen.
5	15	
2	12	
6	16	Ausgehende Kante auf Knoten 8 und 18 mit gleichem *relType*. Dieses Knotenpaar wurde zuvor bereits als äquivalent eingestuft $\Rightarrow$ Kante wird in G_{MCS} aufgenommen.
7	17	

Tabelle 4.3: Traversierungsreihenfolge bei Start von Knoten mit ID 3

Daraus ergibt sich der gemeinsame Subgraph G_{MCS} wie in Abb. 4.35 dargestellt.

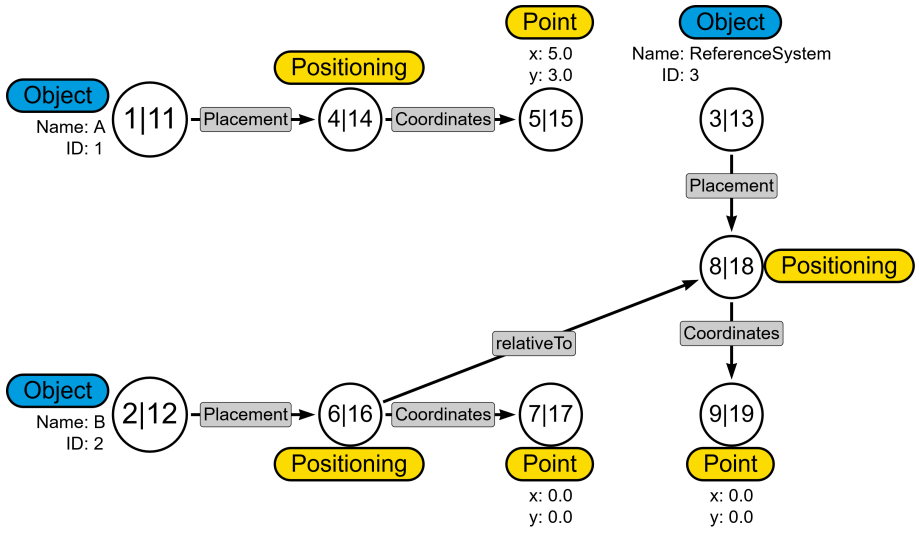


Abbildung 4.35: Gemeinsamer Subgraph G_{MCS} für Traversierungsstart in Knoten mit ID 3

Daraus leitet sich die gleiche Graphtransformation wie in Abschnitt 4.10.2 ab. Wie zuvor muss die Kante $3 \rightarrow 8$ entfernt und eine neue Kante $14 \rightarrow 16$ eingefügt werden. Die resultierende Transformation ist in DPO-Notation in Abb. 4.34 dargestellt.

4.10.4 Schlussfolgerungen

Die geschilderte Situation zeigt, dass der Ansatz der rekursiven Tiefensuche für beliebige Traversierungsreihenfolgen passende Transformationsregeln ermittelt. Die abzuleitenden Transformationen können aber unterschiedliche Gestalt aufweisen. Sichtbar wird dies durch die Gegenüberstellung der Transformationsregeln, die einerseits aus dem Traversierungsstart bei den Knoten mit ID 1 und andererseits bei Start in Knoten mit ID 2 oder 3 hervorgegangen sind. Während als Resultat im ersten Fall je zwei Knoten und 4 Kanten gelöscht und neu eingefügt werden müssen, folgen aus den ermittelten gemeinsamen Subgraphen in den letztgenannten Fällen lediglich notwendige Modifikationen in zwei Kanten. Die Ursache hierfür sind die unterschiedlichen gemeinsamen Subgraphen G_{MCS}. In allen Fällen wurde aber dennoch eine vollständige Transformation gefunden, die den Ausgangszustand G_{init} in den gewünschten Zielzustand G_{updt} überführt.

Die Tatsache, dass für bei der Ermittlung einer Transformationsvorschrift auf Basis zweier Zustände mehrere Lösungen existieren können, wurde bereits für die textuellen Versionskontrollsysteme erläutert (siehe hierzu Abschnitt 2.5.2). Es ist daher nicht überraschend, dass dieses Phänomen nun auch für den Abgleich zweier Graphen und die anschließende Ermittlung einer passenden Transformationsregel auftritt. Mit den definierten Kriterien zur Knotenäquivalenz und zur Behandlung der Kanten im aktuellen Traversierungsschritt wurde aber ein robuster Ansatz gefunden, der auch die beschriebenen Probleme umfänglich und akkurat löst.

4.11 Einordnung und Zusammenfassung

Dieses Kapitel hat ein umfassendes Verfahren zur Repräsentation von BIM-Modellen sowie zur Ermittlung von Änderungen auf Basis ihrer Graphrepräsentationen eingeführt. BIM-Modelle werden hierfür in LPG-Graphen übersetzt. Diese Darstellungsform wird zur Ermittlung eines gemeinsamen Subgraphs G_{MCS} verwendet. Danach können passende Graphtransformationen abgeleitet werden, die einerseits Knotenattribute in Form von semantischen Modifikationen und andererseits topologische Transformationen mit Hinzufügen und Löschen von Teilgraphen beschreiben. Die Transformationen selbst werden mit den PushOut-, Kontext- und Klebe-Muster beschrieben, wobei diese anschließend in die formale Definition eines *Double-Push-Outs* überführbar sind. Die Beschreibung der Inkremente wurde darüber hinaus derart gestaltet, dass die Umkehrung der Transformationsregel und damit deren inverser Anwendung auf einen bereits aktualisierten Graph möglich wird. Dies ermöglicht folglich, dass Versionen nicht nur in chronologischer Reihenfolge produzierbar sind, sondern auch ein Rücksprung auf eine ältere Version gewährleistet ist.

Das beschriebene Vorgehen ermittelt eine möglichst große, aber nicht in allen Fällen die größtmögliche Übereinstimmung zwischen den beiden untersuchten Versionszuständen. Insbesondere in Fällen, bei denen die verglichenen Modellversionen in ihrer topologischen

Struktur weitgehend unverändert sind, wird der ermittelte gemeinsame Teilgraph G_{MCS} der *maximal* möglichen Größe (im Sinne von als äquivalent eingestufter Knoten und Kanten) nahe kommen. Je nach vorgenommenen Modifikationen kann dieser aber von der größtmöglichen Anzahl an Knoten und Kanten, die zwischen G_{init} und G_{updt} als äquivalent einzustufen sind, abweichen. Diese Überlegungen werden in Abschnitt 7.2.4 nochmals aufgegriffen und vorhandene Limitationen mit weiteren repräsentativen Situationen kritisch eingeordnet. Die vorgestellten Algorithmen ermitteln allerdings unabhängig möglicher weiterer Optimierungen in der Ermittlung von G_{MCS} ein passendes Inkrement und übertragen die Änderungen vollständig.

Im nächsten Kapitel wird die erläuterte Methodik nun in ein vollwertiges Versionskontrollsystem erweitert und weitergehende Anwendungsmöglichkeiten zur Erstellung und Kombination von Modellvarianten mithilfe der entwickelten Inkrementbeschreibungen vorgestellt.

Kapitel 5

Anwendung der entwickelten Methodik im Kontext eines Versionskontrollsystems für BIM Modelle

In diesem Kapitel wird die zuvor eingeführte Methodik zur Ermittlung und Anwendung von Versionsinkrementen in mehreren Anwendungen veranschaulicht. Dafür wird in Abschnitt 5.1 einerseits die Nutzung des Diff-and-Patch Vorgehens für chronologisch aufeinander aufbauende Inkremente und andererseits in Abschnitt 5.3 die Weiterentwicklung in ein Branch-and-Merge-System für divergierende Modellzustände und deren Rekombination betrachtet. Abschnitt 5.4 verbindet beide Aspekte und legt dar, wie sich zukünftig modellbasierte Zusammenarbeit im interdisziplinären Kontext unter Zuhilfenahme der vorgestellten Technologien gestalten und wie neben der Versionsüberwachung einer Disziplin auch mögliche Auswirkungen auf andere Disziplinen kommuniziert werden können. Das Kapitel schließt mit einer Zusammenfassung in Abschnitt 5.5.

5.1 Versionskontrollsystem für chronologisch aufeinander aufbauende Versionsinkremente

In Kapitel 4 wurde bereits dargelegt, wie das Objektgefüge eines BIM-Modells als Graph behandelt wird und wie aus zwei Versionen ein passendes Inkrement ermittelt wird. Weiter wurde erläutert, wie das Inkrement anschließend bei einem Empfänger angewendet werden kann, um dort ebenfalls die aktualisierte Form des Graphs zu erzeugen und diesen sofern notwendig zurück in eine dateibasierte Version zu wandeln. Das bisherige Modell wird dafür als initialer Zustand G_{init} bezeichnet, das aktualisierte als G_{updt}.

Da im weiteren Verlauf mehrere Versionen eines Modells untersucht werden, wird erweiternd der Zeitstempel eingeführt, der als *ts* bezeichnet wird und jedem Graph eine eindeutige Versionsnummer zuweist. Dieser wird als zusätzliches Label an alle Knoten angehängt, die den Graphen des entsprechenden Modells bilden. Wie der Name impliziert, beschreibt der Zeitstempel jenen Zeitpunkt, zu dem das Modell in der vorliegenden Form erstellt beziehungsweise aus dem Autorenwerkzeug exportiert wurde. Darüber hinaus erhält jeder Knoten eines Graphs ein Label, welches die erstellende Disziplin repräsentiert.

Um die zuvor beschriebene Methodik stärker an die bekannte Terminologie moderner Versionskontrollsysteme für textbasierte Daten anzulehnen, wurde die CommandLine-Applikation *ConMan* entwickelt. Die darin verwendeten Bezeichnungen sind an den Begrifflichkeiten des Versionskontrollsystems Git und der zugehörigen Kollaborationsplattform

Github angelehnt. Beide Systeme werden heute weit verbreitet für die Versionskontrolle von Quellcode eingesetzt (siehe hierzu auch Abschnitt 2.5.1). Tabelle 5.1 zeigt eine Gegenüberstellung der Funktionalitäten zwischen dem graphbasierten Versionskontrollsystem *ConMan* und Git.

Befehl	Git	ConMan
add	Fügt eine Datei zur Nachverfolgung hinzu und stellt diese für eine zukünftige Commit-Operation bereit.	Übersetzt ein gegebenes (serialisiertes) BIM-Modell in seine graphbasierte Darstellung und speichert es im lokalen Graphspeicher.
get	–	Übersetzt eine graphbasierte Darstellung eines BIM-Modells zurück in eine dateibasierte Darstellung (z. B. in STEP P21, XML oder JSON-Codierung).
commit	Zeichnet die an den vorgemerkten Dateien vorgenommenen Änderungen auf und speichert diesen Snapshot als Commit.	Ermittelt den maximalen gemeinsamen Subgraph G_{MCS} zwischen den zwei untersuchten Versionen und ermittelt sowie speichert das zugehörige Inkrement.
push	Sendet die lokal erstellten Commits an einen anderen Peer im System (z. B. den Remote-Server).	Sendet die Commits an einen Kollaborations-Hub.
fetch	Überprüft, ob neue Commits bei einem entfernten Peer vorliegen.	Überprüft, ob neue Commits auf einem Hub vorliegen.
pull	Holt Commits von anderen Peers in das lokale Repository und wendet deren Änderungen auf die lokal gespeicherten Dateien an.	Holt Commits von anderen Peers und wendet die Transformationen auf die im lokalen Graphenspeicher vorgehaltenen Graphen an.

Tabelle 5.1: Vergleich der zur Verfügung gestellten Funktionalitäten in *Git* (Chacon, 2009) und dem entwickelten graphbasierten Versionskontrollsystem *ConMan*

Abb. 5.1 illustriert den Aufbau des lokalen Repositories bei einem Endnutzer. Dieses ist in wesentlichen Belangen den zuvor erläuterten Strukturen des Git-Versionskontrollsystems für Textsequenzen angelehnt (siehe hierzu auch Abb. 2.8).

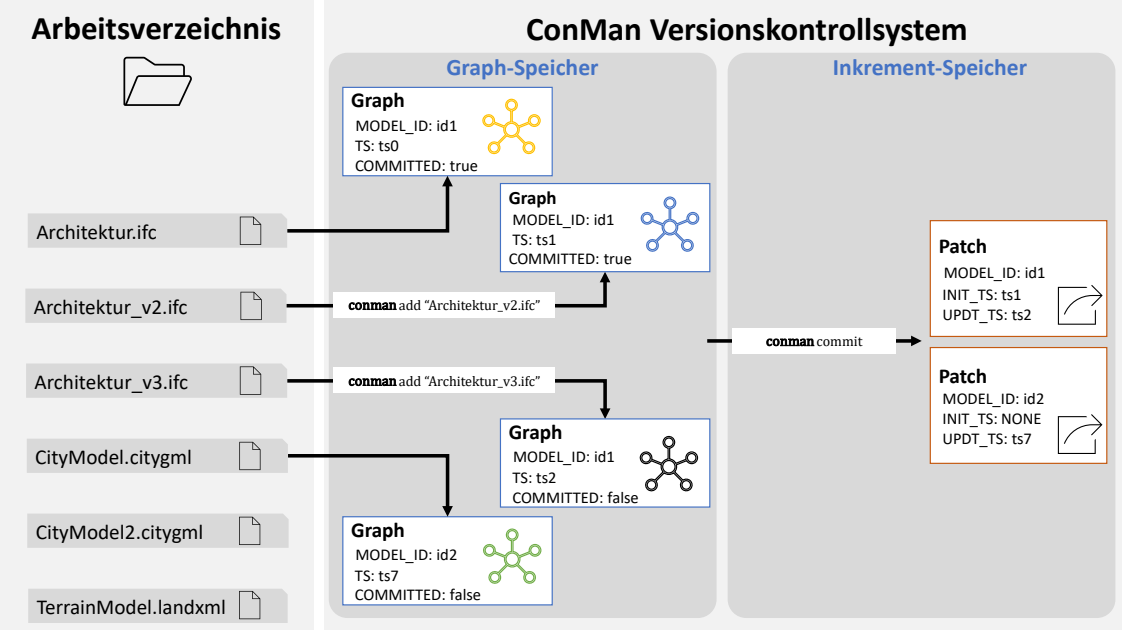


Abbildung 5.1: Bestandteile des Versionskontrollsystems ConMan bei einem Client

Zum Speichern und Interagieren mit den Graphen kommt die Graphdatenbanklösung Neo4j zum Einsatz, die für die prototypische Umsetzung als lokale Installation vorgehalten wird. Die vorgestellten Prinzipien können aber auch mit zahlreichen anderen Systemen erreicht werden, die Funktionalitäten zum Erstellen, Interagieren und Transformieren von LPG-Graphen bereitstellen.

Der Befehl *add* setzt ähnliche Funktionalitäten wie Git um. Bei der Prozedur, die mit dem *commit*-Befehl gestartet wird, werden streng genommen allerdings eher Grundprinzipien aus CVN und Subversion verwendet. Wie in Abschnitt 2.5 dargelegt, speichert Git nicht explizite Inkremente, sondern überwacht alle Dateien und hält für all diese Änderungen beziehungsweise die Verweise auf die zuletzt veränderte Version fest.

Die Verwendung der Versionskontrollmethodik in einem verteilten Versionskontrollsystem mit mehreren Nutzern ist in Abb. 5.2 illustriert. Die Befehle *push*, *fetch* und *pull* synchronisieren die Commits des Inkrementspeichers mit dem zentralen Hub. Dieser bildet das Herzstück des Kollaborationssystems und dient in erster Linie zur Speicherung der eingehenden Commits sowie deren Auslieferung an alle Beteiligte. Er verfügt dabei ebenso wie die lokalen Versionskontrollsysteme über einen Graph- und Inkrementspeicher. Um bestehende Funktionalitäten bisheriger CDE-Plattformen ebenfalls erfüllen zu können, werden im Graphspeicher die aktuellen Modellversionen als Graph vorgehalten und bei Bedarf in serialisierter Form ausgeliefert. Zudem ist vorgesehen, dass der Hub wesentliche Funktionen im Bereich des Nutzer- und Rechtemanagements implementiert. Äquivalent zu den heutigen Möglichkeiten der automatisierten, serverseitigen Kompilierung und des Quellcode-Testens in Plattformen zur Verwaltung von Software-Quellcode können perspektivisch weitergehende Automatisierungen in den Hub integriert werden. Denkbar ist beispielsweise, dass der Hub alle gepushten Commits auf die serverseitig vorgehaltenen Graphen anwendet und anschließend Modellprüfungen anhand vorgegebener Prüfroutinen

durchführt. In einer Minimalvariante wird aber mindestens eine Benachrichtigungsfunktion vorgesehen. Diese soll ermöglichen, weitergehende Aktionen an die Synchronisation von Inkrementen mit der *Push*- und *Pull*-Funktion zu verknüpfen. Gestartet werden können mit eingehenden Benachrichtigungen beispielsweise die Durchführung geometrischer Kollisionschecks über die Generierung von Checklisten bis hin zur automatisierten Aktualisierung verknüpfter BIM-Modelle. Weitergehende Aspekte zu solchen Automatisierungen und den damit verbundenen Herausforderungen erläutert Abschnitt 5.4.

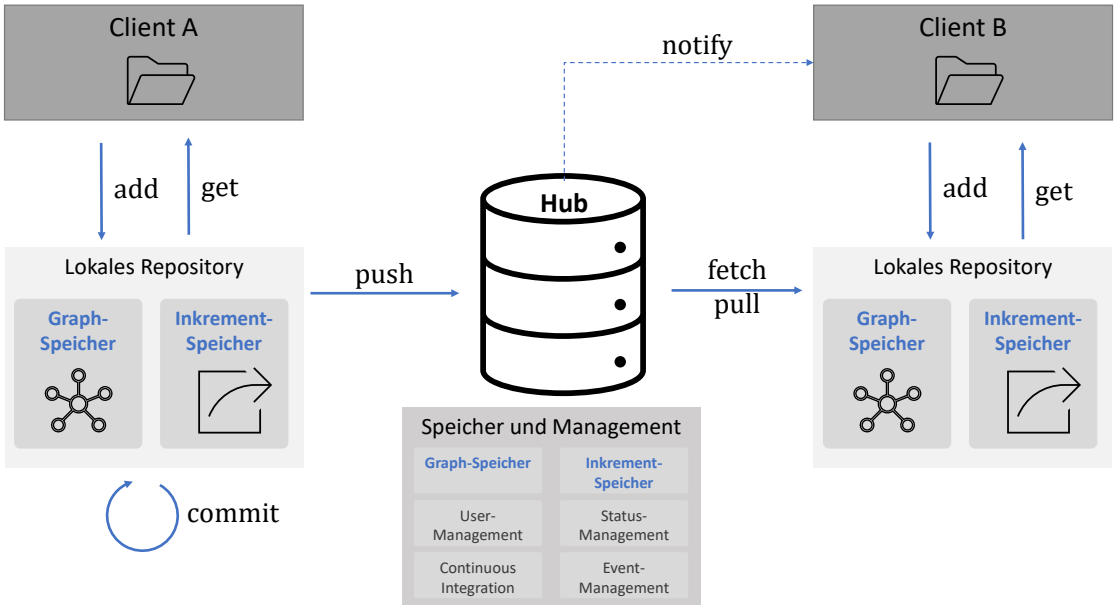


Abbildung 5.2: Versionskontrollsystem mit mehreren Beteiligten sowie einem zentralen Hub, der die Verteilung der Versionsinkremente verantwortet

Vorgesehen ist eine explizite Synchronisierung, die nur auf Wunsch des Nutzers hin ausgelöst wird. Die Möglichkeit zur asynchronen Zusammenarbeit bleibt erhalten, wobei der Informationsaustausch nur zu expliziten Zeitpunkten erfolgt und aktiv durch die Nutzer initiiert werden muss. Die Zeitpunkte zum Austausch von Inkrementen zwischen Nutzer und dem zentralen Koordinations-Hub können projektspezifisch gewählt werden. Möglich sind hochfrequente Synchronisationen im Sekunden-, Minuten- oder Stundenintervall bis zu weniger häufigen Synchronisationen im Wochen- oder Monatsintervall. Es ist jedoch erforderlich, die administrativen und verwaltungstechnischen Aspekte des Projektmanagements zu berücksichtigen, die in dieser Arbeit nicht weiter verfolgt werden.

5.2 Umgang mit Konflikten und divergierenden Versionen in Git

Neben der chronologischen Abfolge von Änderungen verfügen zahlreiche Versionskontrollsysteme auch über Verfahren, mehrere Versionen parallel zu entwickeln und diese bei Bedarf zu vereinigen. Um solche Funktionalitäten im vorgestellten Versionskontrollsystem für BIM-Modelle ebenfalls umsetzen zu können, wird im Folgenden zuerst das Vorgehen

in *Git* erläutert und anschließend die Umsetzung von Mechanismen für divergierende Modellversionen beleuchtet.

In Ergänzung zu den vorgestellten Funktionalitäten ermöglicht es Git, gleichzeitig mehrere Zweige (im Englischen *Branches*) zu verwalten und damit unabhängige Entwicklungen verschiedener Versionsstränge zuzulassen (Chacon, 2009, S. 63–104). Ein Branch ist im Wesentlichen eine Abfolge mehrerer Commits, die es ermöglicht, Änderungen an einem Projekt isoliert zu versionieren, ohne andere Entwicklungen oder Alternativen zu beeinträchtigen. Ein Commit hat dabei entweder gar keinen Vorgänger (Projektinitialisierung), einen Vorgänger-Commit innerhalb eines Branches und mehrere Vorgänger-Commits, sofern der Commit aus der Zusammenführung (engl. *Merge*) zweier Branches resultiert. Diese Arbeitsweise hat sich insbesondere in der Softwareentwicklung etabliert und ermöglicht es einem Entwickler, parallel an verschiedenen Aspekten im Quellcode zu arbeiten und zu diskreten Zeitpunkten alle vorgenommenen Änderungen wieder in eine konsolidierte Version zusammenzuführen.

Abb. 5.3 illustriert eine typische Situation mit *Main*- und *Dev*-Zweig sowie zwei *Feature-Branches*. Ergänzend ist zusätzlich ein sogenannter *Hotfix* dargestellt. Solche direkten Commits auf dem *Main*-Branch werden in der Regel genutzt, um dringende Änderungen in der ausgelieferten Version vorzunehmen.

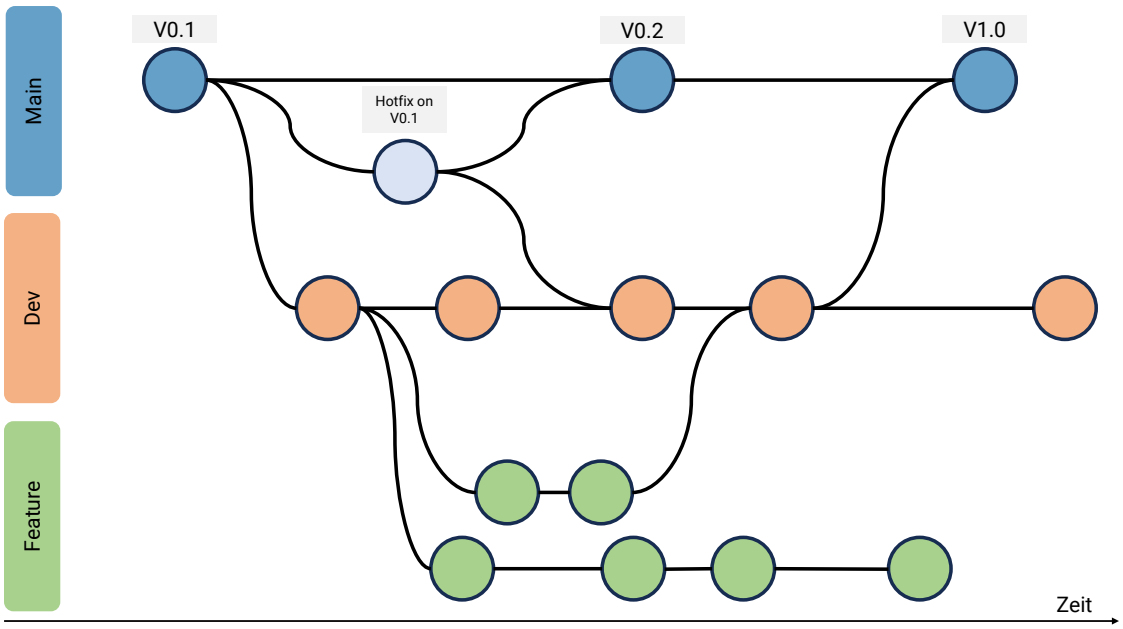


Abbildung 5.3: Beispielhafte Darstellung für die Arbeit mit mehreren Entwicklungszweigen in Git

Die Arbeit mit Zweigen ist ein essentielles Konzept, um Teams zu einer gemeinsamen Softwareentwicklung zu befähigen. Dabei wird in der Regel ein geschützter Hauptzweig (häufig als *Main* oder *Master* bezeichnet) genutzt, der den aktuellen ausgelieferten Stand einer Software beinhaltet. Parallel dazu verläuft ein Entwicklungszweig (auch *Dev* genannt), der im wesentlichen dem Stand des *Main*-Zweiges entspricht, aber auf dem neue Änderungen vor ihrer Freigabe im produktiven Einsatz getestet werden. Erst bei positivem Abschluss

aller zugehörigen Tests werden die vorgenommenen Änderungen des *Dev*-Zweiges in den *Main*-Zweig integriert. Die Entwicklung neuer Funktionalitäten erfolgt darüber hinaus ebenfalls in einzelnen Zweigen, die zumeist als *Feature-Branches* bezeichnet werden. Solche Erweiterungen betreffen in den meisten Fällen viele verschiedene Quellcode-Dateien, die durch das Versionskontrollsystem überwacht werden. Die Verwendung von Branches ermöglicht es Entwicklern, verschiedene Änderungen sauber von einander zu trennen und diese erst nach Fertigstellung in den *Dev*-Zweig zu integrieren. Außerdem bietet es sich an, notwendige Änderungen und Erweiterungen je Feature möglichst präzise zu erfassen, um im Falle auftretender Konflikte zwischen parallel entwickelten Branches sensitiv entscheiden zu können, welche Änderungen den Vorzug erhalten sollen.

Änderungen im *Main*- und *Dev*-Branch vorzunehmen, ist in der Regel nur für autorisierte Nutzer mit höheren Rechten gestattet. Daher wird die Integration von neuen Funktionen beziehungsweise deren zugehöriger Zweige meistens von umfangreichen Prüf- und Freigabeprozessen begleitet, die als *Pull-* oder *Merge-Requests* bekannt sind. Dabei fragt der Autor des *Feature-Branches* den übergeordneten Zweig an, ob vorgenommene Änderungen integriert werden dürfen. Die Entscheidung über die Integration trifft anschließend die verantwortliche Person für den übergeordneten Zweig.

Zur Interaktion mit Branches stehen in Git weitere Befehle zur Verfügung, welche die Funktionen aus Tabelle 5.1 erweitern und in Tabelle 5.2 gegeben sind.

Befehl	Git
branch	Erstellt einen neuen Zweig. Als Argument ist der Zweigname anzugeben.
checkout	Wechselt in den im Argument gegebenen Zweig, um dort Änderungen vornehmen zu können.
merge	Integriert die Commits des als Argument gegebenen Zweiges in den aktuell ausgecheckten Zweig.

Tabelle 5.2: Funktionalitäten in *Git* (Chacon, 2009) zur Erstellung neuer Branches, dem Wechsel zwischen Branches sowie deren erneuter Zusammenführung

Besonderes Augenmerk gilt nun Situationen, bei denen ein überwachter Inhalt in verschiedenen Zweigen auf konkurrierende Art und Weise modifiziert wurde. Betrachtet wird hierfür eine einfache Textdatei namens *file.txt*, die in Algorithmus 5.1 gegeben ist.

Algorithmus 5.1: Beispielhafte Textdatei file.txt zur Erläuterung der Konfliktverwaltung in Git

```
Text.
- 0
- 1
- 2
- 3
```

Die Datei wird als initialer Zustand mit einem Commit als erste Version im *Main*-Branch gespeichert. Anschließend werden von diesem die Branches *Branch A* und *Branch B* eröffnet.

Im ersten Beispiel wird ein Szenario betrachtet, in dem ein automatisches Zusammenführen parallel erstellter Änderungen möglich ist. In *Branch A* wird dafür das Minuszeichen in Zeile 3 in ein Pluszeichen modifiziert und die Änderung als neuer Commit gespeichert. Ein Merge dieser Änderung in den *Main*-Branch ist ohne Probleme möglich, wenn in der Zwischenzeit keine sonstigen Änderungen vorgenommen wurden. In Git wird ein solcher Fall auch als *Fast-Forward-Merge* bezeichnet, da die Commits des zu integrierenden Zweiges lediglich an den Zielbranch angehängt werden müssen.

Parallel zu diesen Modifikationen wird in *Branch B* eine weitere Zeile eingefügt und ebenfalls als Commit gespeichert. Festzustellen ist nun, dass in beiden Branches Änderungen an derselben Datei vorgenommen wurden und diese durch entsprechende Commits festgehalten sind. Ein einfaches Aneinanderreihen der Commits ist nun nicht mehr möglich. Man spricht stattdessen von einem *3-Way-Merge*. Eingeführt wird ein weiterer Commit, der das Verschmelzen explizit beschreibt. Im gewählten Beispiel haben die Inkremente in *Branch A* und *Branch B* auf verschiedene Stellen der überwachten Datei zugegriffen. Daher kann Git beide Änderungen erfolgreich und ohne weitere Nutzerinteraktion in den *Main*-Branch integrieren. Alle beschriebenen Änderungen sind in Abb. 5.4 illustriert.

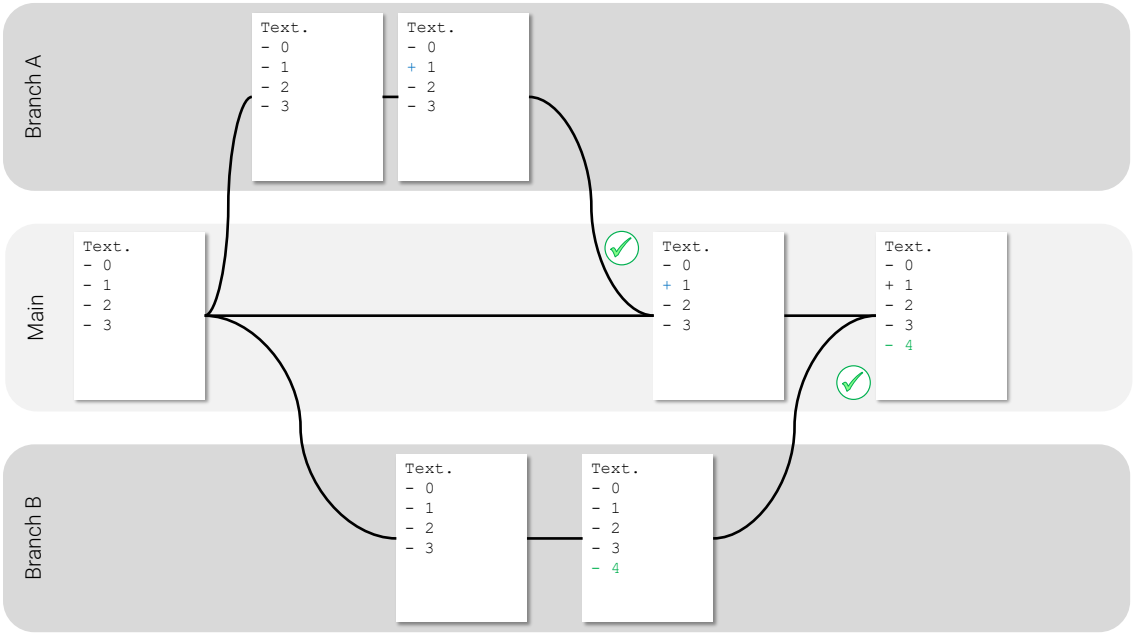


Abbildung 5.4: Erfolgreiches Zusammenführen parallel vorgenommener Änderungen

Das vorherige Beispiel wird nun um konkurrierende Änderungen in den Zweigen erweitert. Die geänderte Situation in ist in Abb. 5.5 dargestellt. Hierzu wurde erneut die in Algorithmus 5.1 dargestellte Textdatei als Ausgangszustand genutzt. In *Branch A* werden in diesem Beispiel die Zeilen 2 und 3 gelöscht, wohingegen in *Branch B* in den gleichen Zeilen lediglich eine Modifikation vorgenommen wurde. Erneut werden zuerst die Ände-

rungen aus *Branch A* in den *Main*-Branch integriert. Zum Konflikt kommt es nun, wenn anschließend der Commit aus *Branch B* in den *Main*-Branch eingefügt werden soll. Dieser kann nur durch explizite Eingaben des Nutzers gelöst werden, der entscheiden muss, welche der konkurrierenden Änderungen übernommen werden sollen.

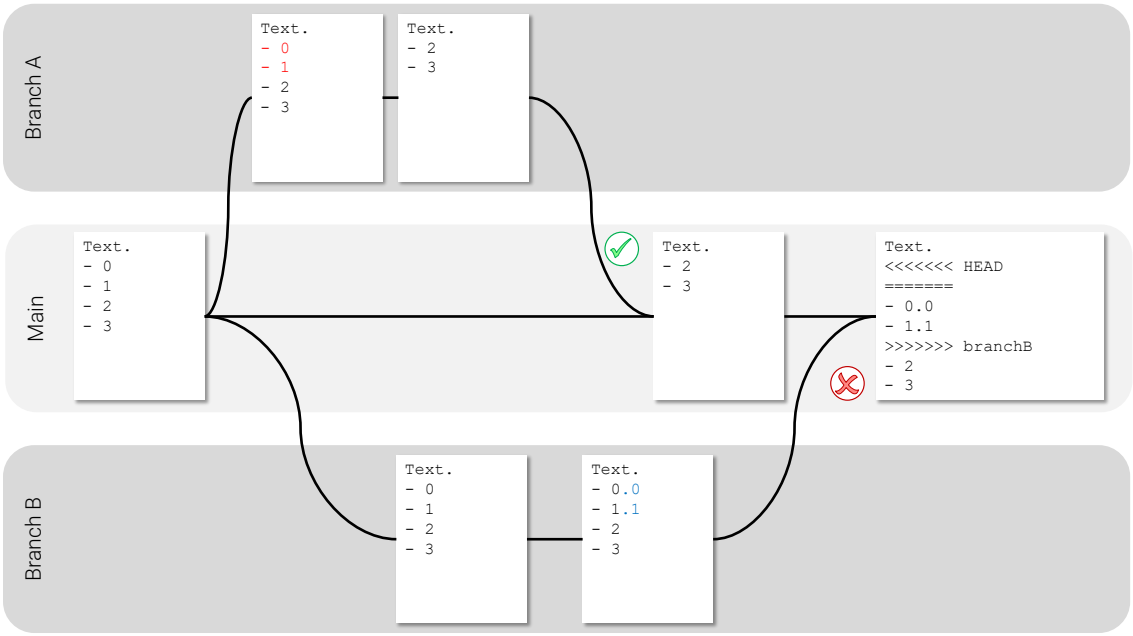


Abbildung 5.5: Konflikt aufgrund konkurrierender Änderungen in versionierter Datei

Eine ähnliche Problematik ist im dritten Beispiel zu beobachten. Hier wird die Datei in *Branch A* gänzlich gelöscht. In *Branch B* werden erneut Modifikationen vorgenommen, die anschließend in den *Main*-Branch integriert werden sollen. Abb. 5.6 illustriert die zugehörige Situation.

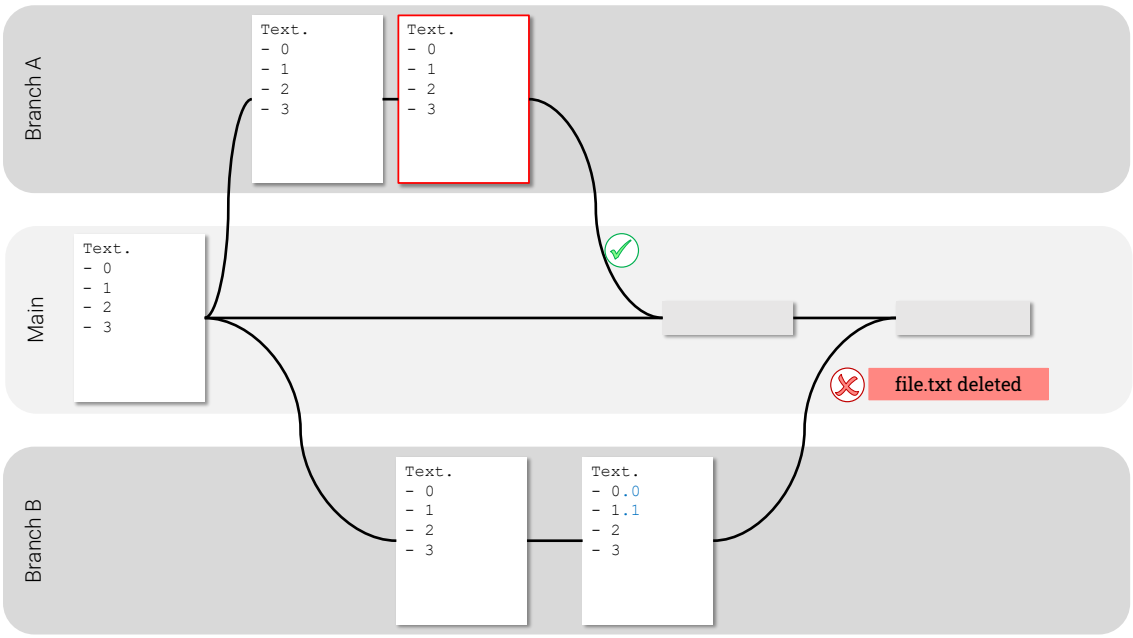


Abbildung 5.6: Konflikt aufgrund Entfernens der Zieldatei aus dem Repository

Da die Zieldatei nach erfolgreichem Merge der Änderungen aus *Branch A* nicht mehr existiert, entsteht ein Konflikt, die den Merge des *Branches B* verhindert. Diese Situation ist nur durch aktives Eingreifen des Nutzers lösbar, der entscheiden muss, ob die gelöschte Datei wieder hergestellt werden soll oder ob die Änderung aus *Branch B* ohne Integration verworfen werden soll. In Algorithmus 5.2 ist die Ausgabe des Befehls `git merge branchB` dargestellt. Als aktiver Branch ist erneut der *Main-Branch* ausgewählt.

Algorithmus 5.2: Git Konsolenausgabe Beispiel 3

```
$ git merge branchB
CONFLICT (modify/delete): file.txt deleted in HEAD and modified in branchB.
    Version branchB of file.txt left in tree.
Automatic merge failed; fix conflicts and then commit the result.
```

Diese Überlegungen verdeutlichen die Komplexität, die die Verwaltung und das Lösen von in Konflikt stehenden Versionen bereits für einfache Textdateien in das System einträgt. Es kann allerdings abgeleitet werden, dass parallel vorgenommene Änderungen an einer Datei dann ohne Konflikte übernommen werden können, wenn die betrachteten Modifikationen unterschiedliche Teile betreffen. Diese Konflikte treten insbesondere in Fällen auf, bei denen gegensätzliche Modifikationen vorgenommen wurden oder die zu modifizierenden Teile aus dem Repository entfernt wurden.

Gleichzeitig bietet das Zusammenführen divergierender Versionen ein hervorragendes Instrument, um Konflikte in transparenter Weise zu lösen und diese entsprechend im Versionskontrollsystem als eigene Commits zu dokumentieren. Abschließend sei nochmals darauf verwiesen, dass Git lediglich syntaktische Änderungen verfolgt. Es kann daher bei der Integration von Zweigen zu Situationen kommen, bei denen Git keine Konflikte erkennt, die repräsentierten Informationen aber nach dem Merge logische Fehler beinhalten.

5.3 Umgang mit divergierenden Varianten von BIM-Modellen

In den in Abschnitt 5.1 dargelegten Erläuterungen zur Versionskontrolle von BIM-Modellen wurde angenommen, dass ein Modellautor immer chronologisch inkrementelle Aktualisierungen vornimmt. Dies bedeutet, dass ein oder mehrere chronologisch aufeinander folgende Inkremente stets auf die (veralteten) Graphrepräsentation des versionierten BIM-Modells auf der Empfängerseite angewendet werden. Zudem wurde angenommen, dass immer nur ein einzelner Modellautor Aktualisierungen für ein bestimmtes BIM-Modell und dessen jeweilige Versionen erstellt.

Heutzutage sind Entwurfsprozesse allerdings höchst iterativ und häufig interdisziplinär, sodass in zahlreichen Situationen mehrere Beteiligte verschiedene Versionen und Varianten eines Modells erstellen und analysieren. Denkbar ist beispielsweise, dass verschiedene Autoren ausgehend von einem Ausgangszustand ein BIM-Modell für unterschiedliche Anwendungsfälle weiterentwickeln. Ähnliche Herausforderungen können darüber hinaus in der Synthese verschiedener Entwurfsentscheidungen entstehen, wenn etwa im Rahmen einer Optimierungsaufgabe mehrere Varianten eines Modells erzeugt und in einer

Simulation ausgewertet werden. Abb. 5.7 zeigt ein Beispiel, bei der ein Ausgangsmodell in unterschiedlichen Aspekten weiterentwickelt wird und anschließend Änderungen verschiedener Varianten in einem konsolidierten Modell zusammengeführt werden.

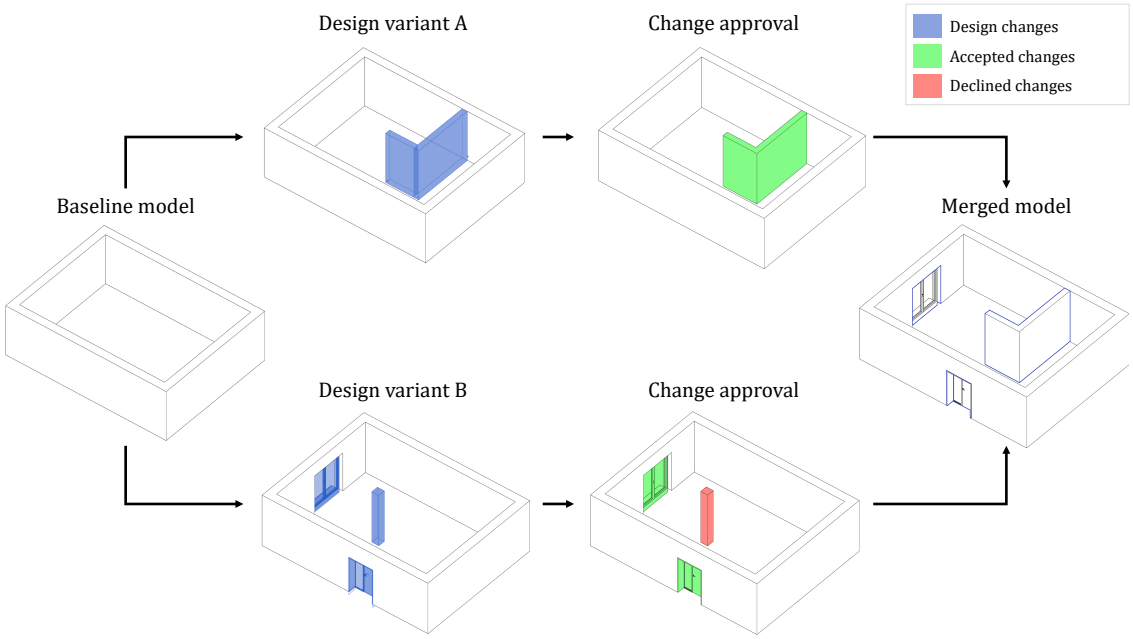


Abbildung 5.7: Fusionieren verschiedener Entwurfsvarianten

Um der skizzierten Komplexität passend zu begegnen, wird das erläuterte Vorgehen zur inkrementellen Versionskontrolle für BIM-Modelle um Mechanismen zur Durchführung von Verzweigungen (*Branching*) und dem späteren Zusammenführen (*Merging*) von divergierender Modellversionen erweitert. Diese Mechanismen sollen im Folgenden ausführlich erläutert und anhand verschiedener anschaulicher Szenarien diskutiert werden. Die vorgestellte Methode folgt weiterhin den etablierten Prinzipien der föderierten BIM-Zusammenarbeit, stattet die beteiligten Parteien jedoch mit zusätzlichen Funktionalitäten aus, um Versionsinkremente, die auf unterschiedlichen Versionen formuliert wurden, wieder in einer gemeinsamen, harmonisierten Modellversion zusammenzuführen.

Dafür sollen die zuvor erläuterten Prinzipien zur Verwaltung gleichzeitiger Entwicklungsstränge aus *Git* adaptiert werden und für die Verwaltung parallel bearbeiteter Varianten von BIM-Modellen genutzt werden. Besonderes Augenmerk muss hierbei auf möglicherweise auftretende Konflikte und Widersprüche in den Modellinformationen gelegt werden. Diese können beispielsweise aufgrund der Bearbeitung derselben Stelle durch mehrere Parteien entstehen oder durch die Veränderung von sich referenzierenden Bestandteilen auftreten. Während ersterer Fall ausschließlich auf der Gegenüberstellung zweier Versionsinkremente und dem Abgleich der durch sie modifizierten Teile identifiziert werden kann, muss für die Detektion fehlerhafter Referenzen zwischen verschiedenen Bestandteilen in der Regel eine Interpretation der versionskontrollierten Datenmenge im Kontext des zugrunde gelegten, spezifischen Datenmodells durchgeführt werden.

Weitergehende Herausforderungen ergeben sich auch in den Überlegungen zur Rekombination divergierender Modellzustände auf Basis ihrer Graphrepräsentationen. Die *Git*-Prinzipien behandeln erneut lediglich textuelle Repräsentationen und nutzen dafür im Wesentlichen Grundprinzipien der Mengentheorie. Im Gegensatz dazu müssen für das Zusammenführen verschiedener Versionen eines BIM-Modells erneut sowohl Knoten als auch die zugehörigen Beziehungen mit in Betracht gezogen werden. Im Weiteren wird daher zuerst erläutert, inwiefern die gegenseitige Abhängigkeit von Graphtransformationen zu beschreiben sind. Anschließend werden basierend auf Fallbeispielen Regeln abgeleitet, in welchen Situationen eine automatische Zusammenführung möglich ist und in welchen Fällen eine Nutzerinteraktion zur Lösung auftretender Konflikte notwendig wird.

5.3.1 Kommutativität von Graphtransformationen

Die Überlegungen zur Verwaltung und Lösung auftretender Konflikte zwischen divergierenden Versionen erfordert die Berücksichtigung weiterer graphtheoretischer Grundlagen. Wie beschrieben wird in *Git* stets jener Nutzer mit der Lösung aufkommender Konflikte konfrontiert, dessen Änderungen im Konflikt mit dem Zielbranch stehen. Gleichzeitig wurde aber auch gezeigt, dass in vielen Fällen eine automatische Lösung gefunden werden kann, wenn zwei Versionen zwar Änderungen an derselben Datei durchgeführt haben, aber die Modifikationen an sich nicht konkurrieren. Um letztgenannte Situation auf die Überlegungen zum Umgang mit Graphen zu übertragen, werden Betrachtungen zur *Kommutativität* von Graphtransformationen relevant. Diese beschreibt die Eigenschaft einer Operation, bei der sich die Reihenfolge der Operanden nicht auf das Ergebnis auswirkt. Eichhoff und Roller (2016) haben sich mit der Kommutativität bei der Anwendung von Graphtransformationsregeln auf einen Arbeitsgraph auseinander gesetzt. Die grundlegende Problematik illustriert Abb. 5.8 und wird auch als *Ableitungskonfluenz* von Graphentransformationsregeln bezeichnet. Untersucht wird dabei die Frage, ob die Reihenfolge, in der zwei Transformationsregeln p_1 und p_2 auf einen Arbeitsgraph G angewendet werden, die Gestalt des resultierenden Graphs G_2 beeinflusst. Die Ableitungskonfluenz besagt dabei, dass der resultierende Graph G_2 unabhängig von der Reihenfolge ist, in der die Regeln angewendet werden.

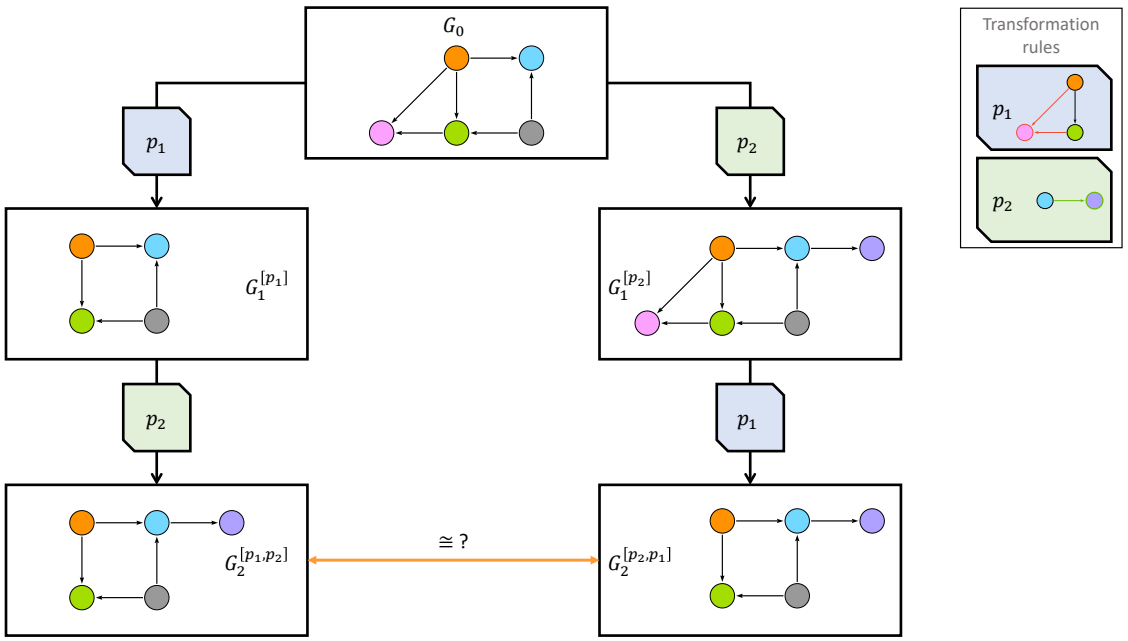


Abbildung 5.8: Kommutativität von Graphersetzungsregeln

Daraus kann abgeleitet werden, dass divergierende Modellversionen dann ohne weitere Nutzerinteraktion zusammengeführt werden können, wenn sich die für diese Versionen ursächlichen Inkremente kommutativ zu einander verhalten. Um die parallele Entwicklung verschiedener Versionen eines Modells zu beschreiben, werden die Überlegungen von Git zur Arbeit mit Branches und dem Merging genutzt. Die Interaktion mit Zweigen folgt dabei den gleichen Befehlen wie in Tabelle 5.2 für Git erläutert.

Betrachtet wird ein BIM-Modell im Zustand V_0, der in einem *Main*-Branch vorliegt. Um mögliche Fälle zu verdeutlichen, werden verschiedene Modifikationen an dieser Modellversion vorgenommen. Wird von einer bestehenden Version ein neuer Zweig erstellt, kann dort das zugehörige Ursprungsmodell V_0 losgelöst von anderen Entwicklungssträngen modifiziert werden. Die jeweiligen Zustände in den Entwicklungszweigen (*Development Branch*) werden mit V_0^A sowie V_0^B benannt, wobei der hochgestellte Buchstabe den entsprechenden Branch bezeichnet und der numerische Index die Versionsnummer repräsentiert. Beide Graphen V_0^A und V_0^B sind untereinander und ebenso zum Ausgangszustand V_0 homomorph und entsprechen topologisch und semantisch dem Ausgangszustand des Modells.

$$V_0 = V_0^A = V_0^B \tag{5.1}$$

Änderungen in den einzelnen Zweigen führen zur Erstellung der Inkremente δ_{ij}^A und δ_{ij}^B.

Betrachtet wird nun die Frage, inwiefern die in den einzelnen Zweigen formulierten Inkremente auf die Version V_0 integriert werden können. Die Anwendung von Inkrementen eines fremden Zweigs auf den aktuellen Zweig wird in Anlehnung an die in *Git* genutzte Terminologie als *Merge* bezeichnet. Die Ausführung eines *Mergings* ist technisch gesehen mit dem Vorgehen des Patchings gleichzusetzen, das in Abschnitt 4.7 beschrieben wurde.

Die wesentlichen Unterschiede bestehen allerdings darin, dass der Arbeitsgraph, auf den ein Inkrement angewendet werden soll, nicht zwingend jener Form entspricht, die bei dem Erstellen des Inkrements als initiale Version G_{init} vorlag. Zusätzlich werden für das *Merging* alle Inkremente eines Branches seit dem letzten Merge oder seit dem Branching berücksichtigt und auf den gewünschten Zielgraph angewendet. Abb. 5.9 beschreibt schematisch eine entsprechende Situation mit einem Hauptzweig (bezeichnet als *Main Branch*) und zwei Entwicklungszweigen (bezeichnet als *Development Branch A* und *Development Branch B*).

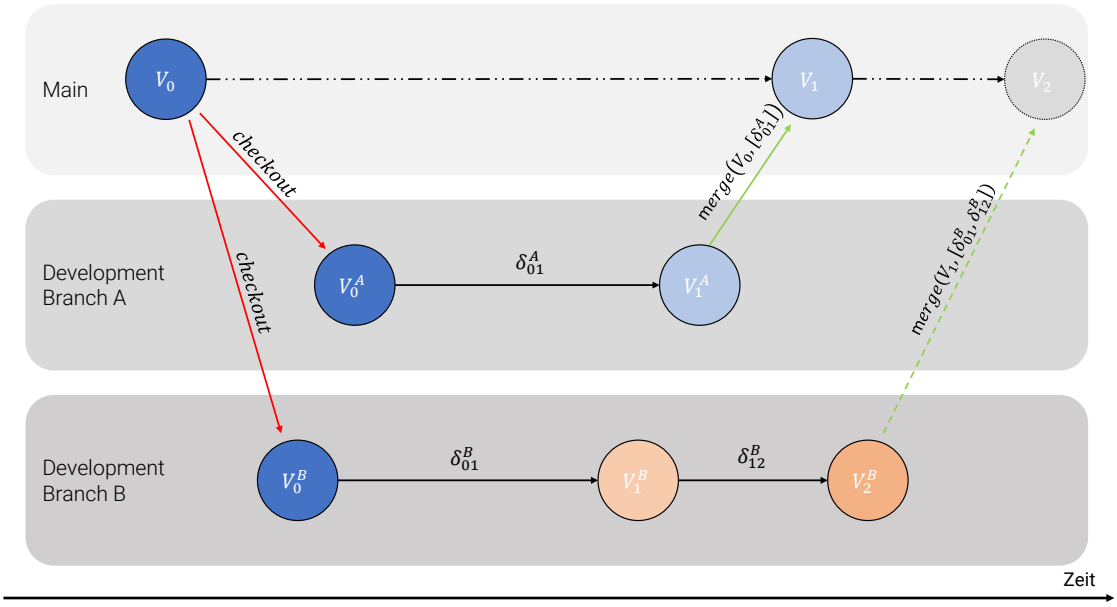


Abbildung 5.9: Konzeptionelle Übersicht des Verzweigens und Zusammenführens divergierender Versionen eines BIM-Modells

Ausgehend von dieser Situation wird nun zuerst die *Merge*-Operation des im Entwicklungszweig formulierten Inkrements δ_{01}^A in den Main-Branch betrachtet. Die Indices bezeichnen dabei die Aktualisierung des überholten Modells 0 auf den aktualisierten Versionszustand 1, der als V_A^1 bezeichnet wird. Da die Versionen V_0 und V_0^A zueinander homomorph sind, kann das Inkrement ohne Einschränkung auch auf V_0 angewendet werden, da in der Zwischenzeit keine sonstigen Änderungen im Main-Branch vorgenommen wurden.

Währenddessen wurden in Branch B zwei Inkremente δ_{01}^B und δ_{12}^B formuliert. Von Interesse ist nun die Frage, ob und unter welchen Randbedingungen diese im Branch B formulierten Inkremente auf die Version V_1 im Main-Branch angewendet werden können. Die Version V_1 wurde im Vergleich zur Ursprungsversion des Branches B bereits durch die Anwendung des Inkrements δ_{01}^A modifiziert.

Zur weitergehenden Illustrierung des Problems zeigen die kommenden Abschnitte praktische Beispiele möglicher Modelländerungen auf. Im Anschluss wird aus diesen Erkenntnissen eine Systematik abgeleitet, unter welchen Bedingungen Inkremente rekombiniert werden können.

5.3.2 Mögliche Szenarien für das Zusammenführen divergierender Modellstände

Die folgenden Szenarien behandeln verschiedene Versionen eines einfachen BIM-Modells, das in seinem Ausgangszustand eine Wand und eine Tür enthält. In allen Fällen werden zwei Zweige mit den Namen *Design Development Branch A* und *Design Development Branch B* durch die Nutzung der Checkout-Version genutzt. Unter Verwendung des Konzepts des Checkouts verfügen beide Zweige zu Beginn über denselben Modellzustand.

Fall 1: Automatisches Zusammenführen

Im ersten Szenario wird die in Abb. 5.10 dargestellte Situation untersucht. Im Zweig A wurde ein neues Fenster links neben der Tür hinzugefügt. Im Zweig B hingegen wurde rechts von der Tür ein neues Fenster eingefügt. Beide Komponenten benötigen die Wand als Basiselement und nutzen das Projektkoordinatensystem zur Verortung im dreidimensionalen Modellraum.

Bei der detaillierten Analyse der zugehörigen Inkremente wird ersichtlich, dass beide Einfügeoperationen aus einem PushOut-Teil bestehen, der das neu eingefügte Fenster, seine Platzierung, die geometrische Darstellung, den erforderlichen Freiraumkörper und zugehörige semantische Informationen darstellt. Das Kontext-Muster gibt die Wand und andere Modellinformationen an, die erforderlich sind, um das korrekte Einfügen des Fensters in das Modell sicherzustellen. Dies umfasst Aspekte wie den geometrischen Kontext, das Referenzkoordinatensystem oder die korrekte Zuordnung von Einheiten. Dementsprechend verbindet das Klebe-Muster den neu eingefügten Teilgraph mit den diversen bereits im Graphen vorhandenen Informationen.

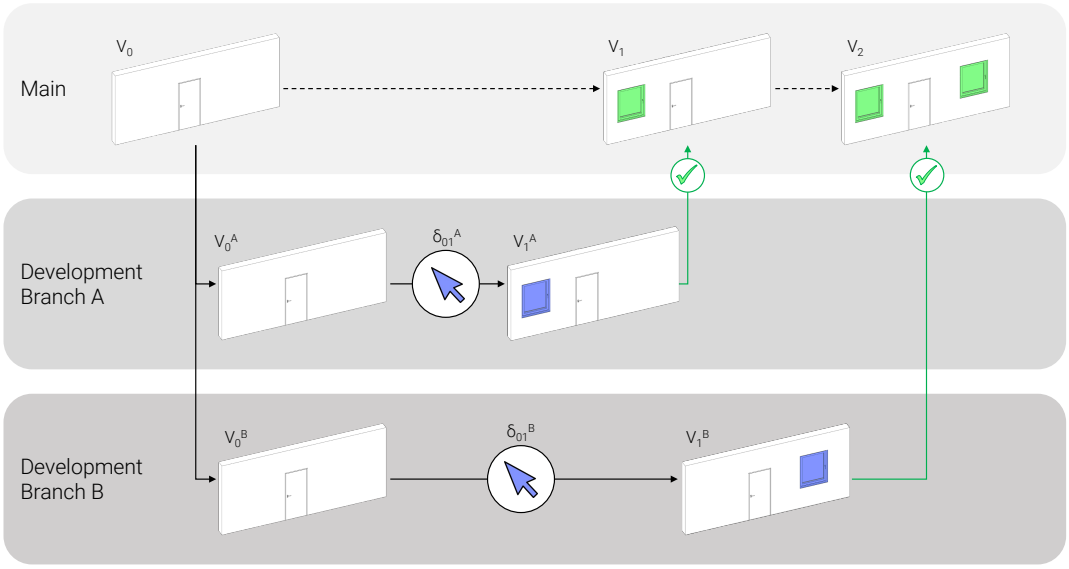


Abbildung 5.10: Fall 1: Automatisches Zusammenführen zweier Modellversionen möglich

Abb. 5.11 veranschaulicht die Patch-Anwendung qualitativ. Die in grün gezeichneten Knoten zeigen die neu eingefügten Teilgraphen, während die roten und grünen Kanten die Kanten des Klebe-Musters darstellen. Wie erwartet, erfordern beide Operationen, dass spezifische Knoten im Arbeitsgraph existieren, so dass die eingefügten Teilgraphen entsprechend und eindeutig mit dem vorhandenen Knoten verbunden werden können.

Betrachtet wird nun, ob beide Inkremente δ_{01}^{A} und δ_{01}^{B} auf die Ausgangsversion V_0 angewendet werden können. Wie zuvor kann das Inkrement δ_{01}^{A} aus Branch A ohne Einschränkung auf V_0 angewendet werden, da in beiden Zweigen der gleiche Ausgangszustand für die Formulierung des Inkrements genutzt wurde. Das Einfügen des Fensters führt vorrangig zu einer Erweiterung des Graphs und erzeugt neue Kanten zwischen dem neu eingefügten PushOut-Muster und den bestehenden Knoten, die im Kontext-Muster spezifiziert wurden. Analysiert man nun die Modifikationen, die durch das Inkrement δ_{01}^{B} vorgenommen werden sollen, so ist eine vergleichbare Charakteristik wie in den Änderungen zu erkennen, die in δ_{01}^{A} enthalten sind. Auch hier wird ein neuer Teilgraph durch das PushOut-Muster hinzugefügt und mit bestehenden Knoten des bisherigen Graphs verbunden.

Zu erwähnen sei an dieser Stelle, dass beide Inkremente ähnliche Kontext-Muster nutzen, um die notwendigen Informationen im bestehenden Graph zu spezifizieren. Gleichzeitig ist im gegebenen Szenario keine gegenseitige Beeinflussung der beteiligten Kontext-Muster zu erwarten, da beide Inkremente vor allem neue Teilgraphen einfügen und diese mit bestehenden Strukturen verbinden. Die eingefügten Teilgraphen repräsentieren dabei die neu hinzugefügten Modellkomponenten mit Informationen über deren geometrische Gestalt, zugehörige Materialien sowie ihrer Platzierung innerhalb des Modellraums. Daraus wird abgeleitet, dass das Inkrement δ_{01}^{B} auf die Version V_1 angewendet werden kann und Version V_2 resultiert. Diese Version beinhaltet anschließend die Wand, die Tür und beide Fenster.

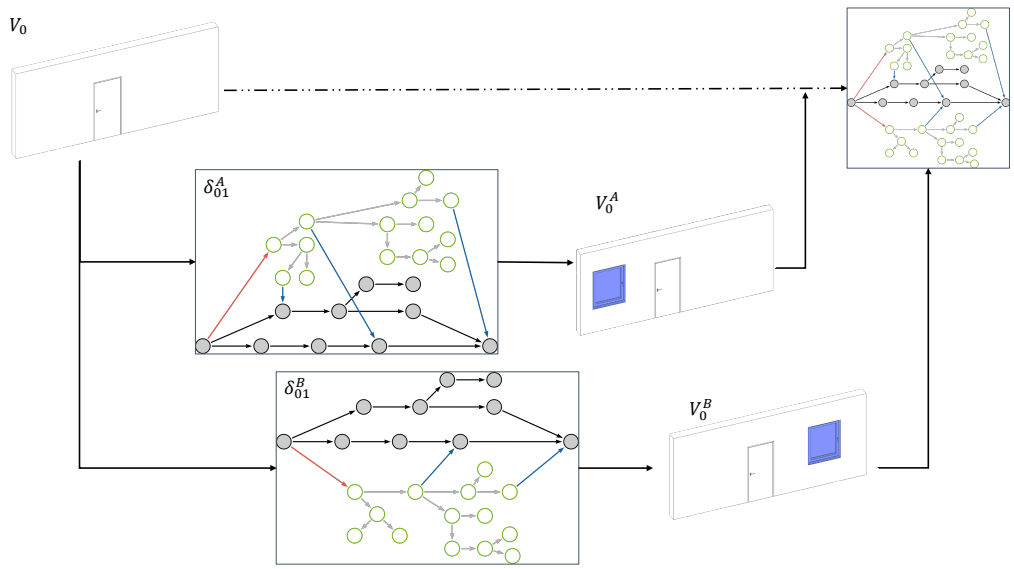


Abbildung 5.11: Qualitative Darstellung der topologischen Modifikationen der in Fall 1 behandelten Inkremente

Als Besonderheit des skizzierten Falls sei ergänzend angemerkt, dass es bei der Anwendung beider Inkremente unter Umständen kleine Anpassungen in den semantischen Informationen der Kanten in den Klebe-Mustern notwendig sind. Wie in Abschnitt 4.4 erläutert, wird das Attribut *listItem* verwendet, um eine Liste mehrerer Assoziationen zu speichern. Im dargelegten Fall kann nun der Fall eintreten, dass ein Inkrement eine Listenposition beansprucht, die bereits vergeben wurde. Um hier die vollständige Integrität der Informationen zu gewährleisten, muss hier eine Anpassung der entsprechenden Kantenattribute bei der Anwendung des Inkrements vorgenommen werden. Abb. 5.12 illustriert das Vorgehen und die vorgeschlagene Lösung.

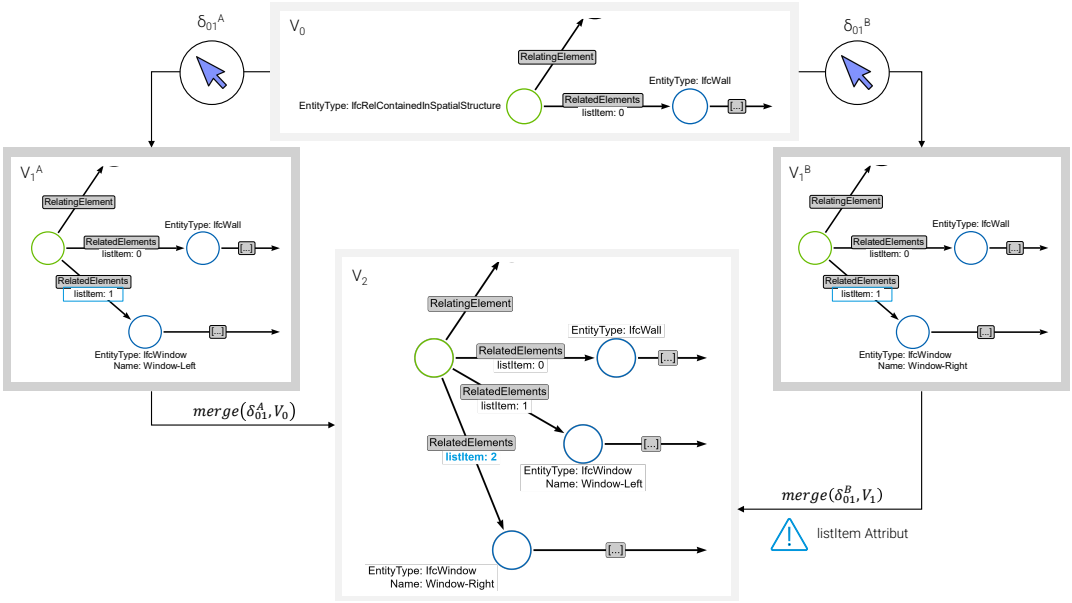


Abbildung 5.12: Modifikation der Kanten im Klebemuster, sofern die gewünschte Listenposition bereits vergeben ist

Fall 2: Erforderliche Nutzerentscheidung vor dem Zusammenführen

In diesem Szenario werden nun zwei semantische Änderungen betrachtet, die die Position der Tür im Modell verändern. Im Branch A wurde veranlasst, die Tür um -1.20 zu verschieben. Diese Modifikation wird im Inkrement δ_{01}^{A} gespeichert und bewirkt, dass die Tür nach Anwendung des Inkrements die Position $(2.05, 0.0, 0.0)$ besitzt. Im Gegensatz dazu wurde die Tür im Branch B um $+1.05$ verschoben, was in einem entsprechenden Inkrement δ_{01}^{B} und einer Position der Tür $(4.30, 0.0, 0.0)$ resultiert. Die resultierenden Modellzustände sind in Abb. 5.13 dargestellt.

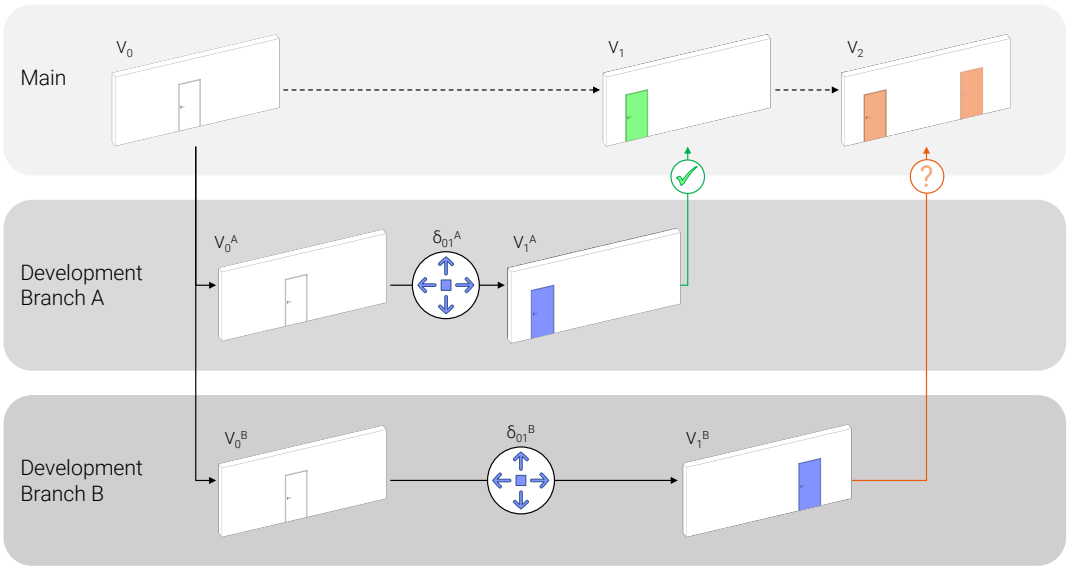


Abbildung 5.13: Fall 2: Erforderliche Nutzerentscheidung vor dem Zusammenführen

Untersucht wird nun, ob das in Branch B erstellte Versionsinkrement δ_{01}^{B} in den Modellzustand V_1 integriert werden kann, der sich zuvor aus dem Integrieren des Inkrements δ_{01}^{A} aus Branch A ergeben hat. Im Gegensatz zum ersten Szenario führen die angewendeten Änderungen zu semantischen Modifikationen von Knoten, die in allen Zweigen und betrachteten Versionen vorhanden sind. Demnach handelt es sich um rein semantische Modifikationen, bei denen die Knotenattribute, die die Position der Tür angeben, verändert werden. Abb. 5.14 zeigt den Pfad, zu dem eine Passung erzielt werden muss, um mit dem Inkrement δ_{01}^{B} den Knoten D zu modifizieren. Um den zu ändernden Knoten eindeutig zu beschreiben, wird ein Pfad verwendet, der den Primärknoten der Tür inklusive des eindeutigen Attributs *GlobalId* beinhaltet. Zudem enthält das Muster die Koordinatenwerte, die die Türposition ausgehend von der für das Inkrement δ_{01}^{B} verwendeten Initialversion V_0^A angibt. Da das Inkrement auf Basis der Version V_0 beziehungsweise V_0^B erstellt wurde, handelt es sich bei dem Attributwert im Attribut *Coordinates* um die Position der Tür im Ausgangszustand.

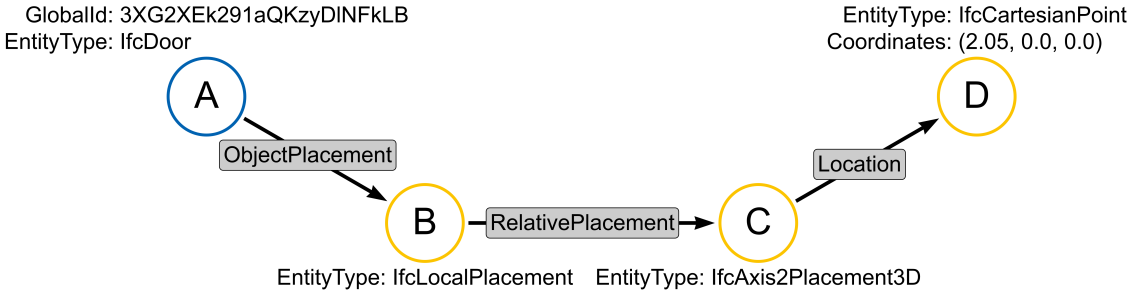


Abbildung 5.14: Eindeutiger Pfad des Inkrements δ_{01}^{B} zur Aktualisierung des Attributes *Coordinates*

Obwohl die relevante Information über der Türposition immer noch topologisch an derselben Stelle im Graph vorhanden ist, wird die Passung des im δ_{01}^{B} gegebenen Musters fehlschlagen, wenn zuvor bereits δ_{01}^{A} auf die Modellversion V_0 angewendet wurde. Damit befindet sich die Tür nicht mehr an der Position $(3.25, 0.0, 0.0)$, sondern wurde an die Position $(2.05, 0.0, 0.0)$ verschoben. Die zugehörige Notation zeigt Algorithmus 5.3.

Algorithmus 5.3: Cypher-Statement zur Passung des eindeutigen Pfads mit Angabe des Initialwerts

```
MATCH p1 =
(A:PrimaryNode{EntityType: "IfcDoor", GlobalId: "3XG2XEk291aQKzyDlNFkLB"})
 -[:rel{rel_type: "ObjectPlacement"}]->
(B:SecondaryNode{EntityType: "IfcLocalPlacement"})
 -[:rel{rel_type: "RelativePlacement"}]->
(C:SecondaryNode{EntityType: "IfcAxis2Placement3D"})
 -[:rel{rel_type: "Location"}]->
(D:SecondaryNode{EntityType: "IfcCartesianPoint",
                 Coordinates: "(3.25, 0.0, 0.0)"})
RETURN EXISTS(p1)
```

Der Pfad $p1$ für die semantische Transformation im Inkrement δ_{01}^{B} wird im Graph V_1 keine Passung erzielen, wenn das *Coordinates*-Attribut mit im entsprechenden Cypher-Statement spezifiziert wird. Offenkundig stehen die beiden Inkremente aus den Entwicklungszweigen A und B im Widerspruch zueinander. Die Analyse der semantischen Modifikationen für die Inkremente δ_{01}^{A} und δ_{01}^{B} zeigt aber, dass diese aus topologischer Sicht homomorph zu einander sind und lediglich der exakte Koordinatenwert (in den Mustern als *Coordinates* bezeichnet) an dem Knoten D eine erfolgreiche Passung des Kontextmusters aus δ_{01}^{B} in V_1 verhindert.

Die Erkenntnis, dass die beide Pfade Ähnlichkeiten hinsichtlich ihrer topologischen Struktur aufweisen, können nun für weitergehende Überlegungen genutzt werden. Aus der Homomorphie zwischen den Kontext-Mustern in $\delta_0 1^A$ und $\delta_0 1^B$ folgt, dass beide Inkremente den gleichen Knoten im Graph modifizieren. Demnach ist es möglich, mithilfe einer reduzierten Form des Pfades den betroffenen, zu modifizierenden Knoten eindeutig in allen Versionen aufzufinden. Konkret muss im beschriebenen Szenario das Knotenattribut *Coordinates* mit dem gewünschten Wert $(3.25, 0.0, 0.0)$ aus der Formulierung des Musters entfernt werden. Diese Abwandlung ist in Algorithmus 5.4 in Cypher dargestellt.

Algorithmus 5.4: Cypher-Statement zur Passung des eindeutigen Pfads in reduzierter Form

```
MATCH p2 =
(A:PrimaryNode{EntityType: "IfcDoor", GlobalId: "3XG2XEk291aQKzyDlNFkLB"})
 -[:rel{rel_type: "ObjectPlacement"}]->
(B:SecondaryNode{EntityType: "IfcLocalPlacement"})
 -[:rel{rel_type: "RelativePlacement"}]->
(C:SecondaryNode{EntityType: "IfcAxis2Placement3D"})
 -[:rel{rel_type: "Location"}]->
(D:SecondaryNode{EntityType: "IfcCartesianPoint"})
RETURN EXISTS(p2)
```

Das nun spezifizierte Muster $p2$ wird eine erfolgreiche Passung im Graph V_1 erzielen. Sofern die vorgestellte Reduzierung der semantischen Informationen in den Musterbeschreibungen notwendig ist, muss allerdings stets eine Entscheidung durch den Nutzer erfolgen, um den auftretenden Konflikt entsprechend zu lösen. Ein analoges Vorgehen ist auch für topologische Modifikationen vorzusehen, sofern hier die Passung der Kontext-Muster fehlschlägt.

Sofern eine homomorphe Abbildung zwischen den Kontext-Mustern der konkurrierenden Inkremente aus topologischer Sicht existiert, kann der entstandene Konflikt durch eine Nutzerentscheidung durch das Versionskontrollsystem gelöst werden. Die notwendige Eingabe durch den Nutzer beschränkt sich in diesem Fall auf die Auswahl des gewünschten Zielzustands (im Beispiel also die Frage, welche Koordinaten für die Positionierung der Tür in der Modellversion V_2 genutzt werden sollen).

Fall 3: Fehlschlagendes Zusammenführen

In einem dritten Szenario wird das zweite Szenario modifiziert, so dass die Tür im Branch A entfernt wird. In Branch B wird weiterhin die beschriebene Verschiebung der Tür nach rechts behandelt und ein entsprechendes Inkrement formuliert.

Aufgrund der in Branch A verwendeten Version V_0 beziehungsweise des ausgecheckten Zweiges mit Version V_0^A ist das Zusammenführen des Inkrements δ_{01}^A in den Main-Zweig ohne Einschränkungen möglich. Soll jedoch die Positionsänderung, die im Branch B vorgenommen und im Patch δ_{01}^B erfasst wurde, in den Main-Zweig übernommen werden, wird die Anwendung des Inkrements auf V_1 scheitern, da die Tür im Modellzustand V_1 nicht mehr verfügbar ist.

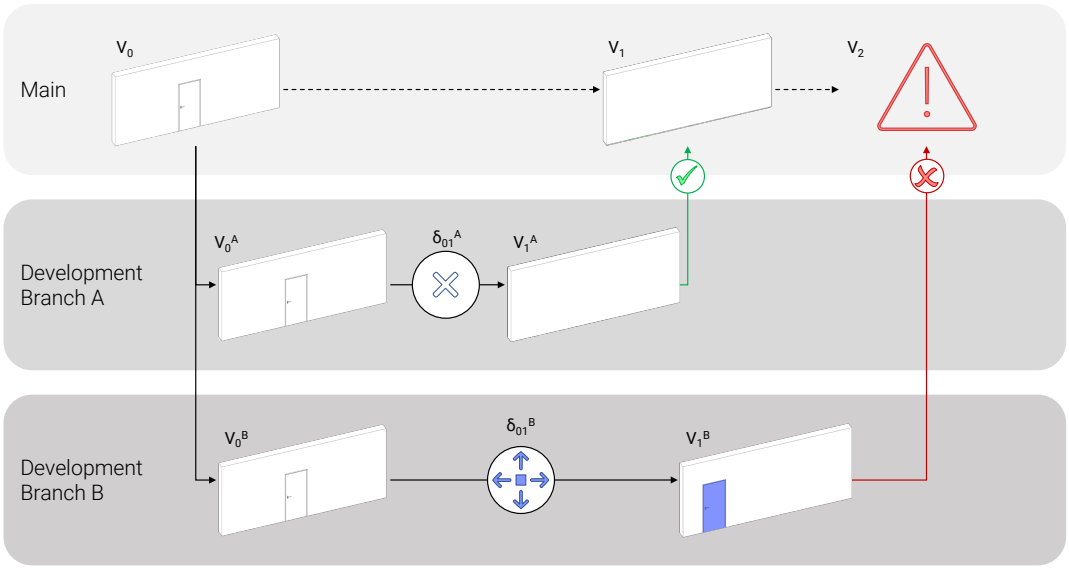


Abbildung 5.15: Fall 3: Automatisches Zusammenführen aufgrund konträrer Inkremente nicht möglich

5.3.3 Abgeleitete Kriterien für die Konfliktlösung

Angesichts der oben genannten Szenarien wird nun eine formale Beschreibung abgeleitet, unter welchen Bedingungen eine Zusammenführung von divergierenden Modellzuständen ohne Konflikte durchgeführt werden kann. Des Weiteren sollen die Fragestellungen aus Fall 2 generalisiert werden, sodass eine einfache Benutzerentscheidung bestimmte Arten von Konflikten auflösen kann. Abb. 5.16 bietet einen Überblick unter welchen Bedingungen das Zusammenführen möglich ist oder im Falle von Konflikten die Zusammenführung durch eine einfache Nutzerentscheidung aufgelöst werden kann. In allen anderen Fällen ist eine umfassendere Analyse erforderlich, um die kollidierenden Änderungen in verschiedenen Patches zu identifizieren.

Aus diesen Untersuchungen werden daher die folgenden Bedingungen abgeleitet:

- Eine konfliktfreie Zusammenführung von Patches ist möglich, wenn alle im betreffenden Patch spezifizierten Kontext-Muster mithilfe semantischer und topologischer Passung erzielen können.
- Eine Benutzerentscheidung ist erforderlich, wenn die Kontext-Muster keine vollständig semantische, aber topologische Passung erzielen.
- In allen anderen Fällen sind weitere Untersuchungen und Kommunikation mit den Autoren der Patches erforderlich. Mögliche Lösungen sind dabei das Verwerfen vorheriger Änderungen oder deren Erweiterung um ein lösendes Inkrement.

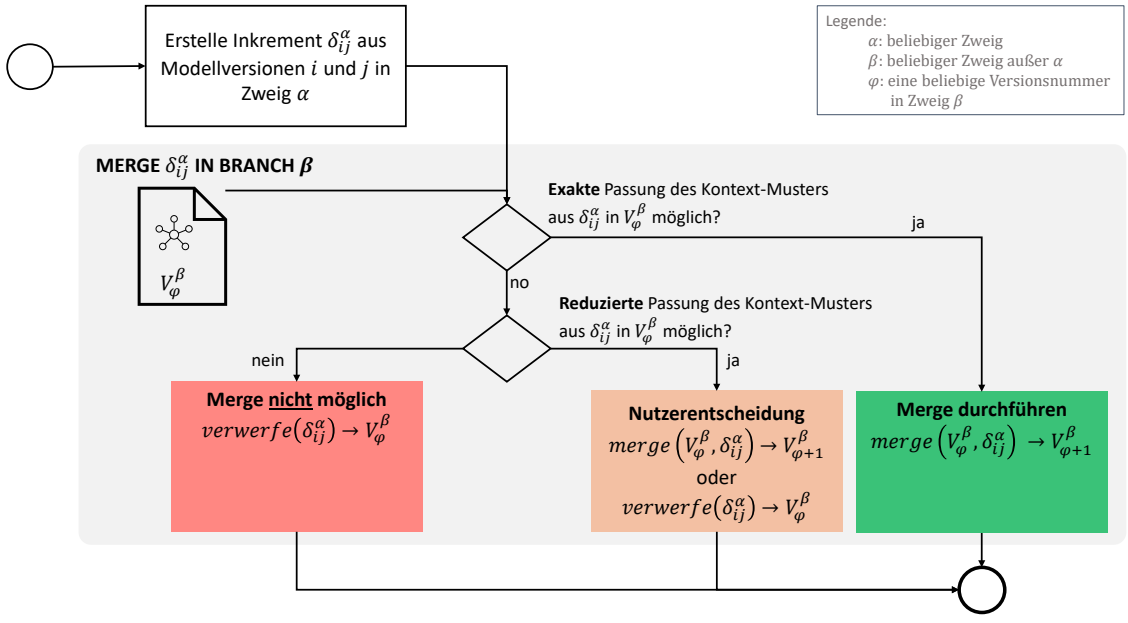


Abbildung 5.16: Kriteriensystem zur Bewertung der Integrierbarkeit parallel erstellter Inkremente in verschiedenen Entwicklungssträngen

Da die Menge der Modifikationen pro Inkrement nicht festgelegt ist, können Kombinationen der genannten Situationen auftreten. Im Allgemeinen steigt die Wahrscheinlichkeit einer

konfliktfreien Zusammenführung mit einer hohen Häufigkeit von Informationsaustausch und Aktualisierungen, die auf eher geringfügige Modifikationen beschränkt sind. Im Gegensatz dazu führen längere Transaktionen typischerweise zu einer größeren Anzahl von Konflikten, die bei der Integration gelöst werden müssen. Ähnlich der Idee einer modellbasierten Versionskontrolle unabhängig vom zugrundeliegenden Datenmodell kann der Zeitpunkt Synchronisierung und Integration ausstehender Inkremente nicht allgemein empfohlen werden. Vielmehr muss die Häufigkeit der durchgeführten Änderungen und die Menge an Modifikationen in jedem Patch an den spezifischen Umständen ausgerichtet werden.

Ein weiterer Aspekt, der umfangreiche Eingaben der Nutzer erfordert, ist die Pflege von Zweigen, die parallel bearbeitet werden. Je weniger Zweige und Inkremente im Versionskontrollsystem zu verwalten sind, desto weniger Varianten existieren im System und müssen letztendlich wieder integriert werden. Aus der Definition der Checkout-Operation wird grundsätzlich sichergestellt, dass jedes BIM-Modell stets eine Vorgängerversion hat. Daher kann der Ursprung von Modellversionen jederzeit zurückverfolgt und veraltete Modellversionen durch die inverse Anwendung der Patches reproduziert werden. Wenn ein vollständig neues BIM-Modell zur Versionsverfolgung hinzugefügt wird, ist eine anfängliche Übertragung seiner Graphrepräsentation erforderlich. In diesem Fall enthält das Delta nur Einfügeoperationen und führt eine triviale Modellkopie durch.

5.4 Interdisziplinäre Koordinationsmodelle und Bewertung von Modelländerungen

Genauer betrachtet werden soll nun nochmals der Graph-Speicher, der als Bestandteil des in Abb. 5.2 gezeigten zentralen Hubs vorgehalten wird. Wie beschrieben werden die Graphen bei jedem Akteur lokal im Versionssystem vorgehalten, um lokale Commits durchzuführen und die zugehörigen Inkremente ermitteln zu können. Der Graph-Speicher im zentralen Hub dient dazu, um im Hub stets alle aktuellen Versionen der verwalteten Modelle als Graph bereitzustellen und diese bei Bedarf auch in serialisierter Form ausliefern zu können (beispielsweise zu Zwecken der Modellvisualisierung oder für Akteure und Systeme, die nicht über einen eigenen Versionsclient angeschlossen sind).

Obwohl jede Disziplin die Verantwortung für ihre Modelle und die darin enthaltenen Objekte trägt, ergeben sich neben Beziehungen zwischen Objekten innerhalb eines Modells auch modellübergreifende Abhängigkeiten zwischen Komponenten, die zusammen später ein gesamtheitliches System in der gebauten Umwelt bilden. Zusätzlich sind Disziplinen häufig auf die Vorarbeiten anderer Domänen für ihre Planungs- und Simulationsaufgaben angewiesen und müssen auf diesen Planungen aufbauen. Beispielhaft sei der Zusammenhang zwischen Tragwerksmodellen und Architekturmodellen genannt. In vielen Fällen leiten sich Tragwerksmodelle und die zu bemessenden Bauteile aus dem Architekturmodell ab. Dieses Wissen wird in der heutigen Praxis aber nicht in die resultierenden Modelle oder in die Kollaborationsplattform integriert, sondern muss immer dann erneut ermittelt wer-

den, wenn die Abhängigkeit zwischen Komponenten verschiedener Modelle relevant wird. Sofern später Änderungen an Bauteilen des Architekturmodells vorgenommen werden, müssen diese in den heute etablierten Verfahren manuell identifiziert und anschließend entsprechend im Tragwerksmodell korrigiert werden.

Weitere Überlegungen können daher angestellt werden, um das Verständnis für Modellveränderungen in einem interdisziplinären Kontext zu stärken und Abhängigkeiten zwischen Objekten verschiedener Disziplinen zu berücksichtigen. Dabei soll bewusst an der asynchronen, föderierten Zusammenarbeit in Teilmodellen festgehalten werden, um die Flexibilität in der Wahl von Autorenwerkzeugen für alle Disziplinen zu erhalten. Angesichts dieser Zielsetzung ist es entscheidend, während des laufenden Entwurfsprozesses verschiedene Darstellungen und Modelle zu verbinden, um die Bewertung von Entwurfsänderungen und deren Auswirkungen auf Modelle anderer Disziplinen zu beschleunigen. Sobald solche Beziehungen eingeführt wurden, können Änderungen viel schneller als in aktuellen BIM-Kollaborationsplattformen bewertet werden.

Im Zentrum der Überlegungen stehen nun verschiedene Handlungsebenen, die durchgeführt werden müssen, wenn Modelländerungen an Modellen auftreten, die bereits in das interdisziplinäre Koordinationsmodell integriert wurden. Daher werden verschiedene Komplexitätsstufen definiert, die ausgelöst werden können, wenn ein Teilmodell innerhalb eines Koordinationsmodells aktualisiert werden soll:

- **Benachrichtigen**: Alle Projektbeteiligte über neue Modelle in einem bestimmten Disziplinmodell informieren.
- **Berichten**: Benutzer darüber informieren, ob Modelländerungen potenzielle Auswirkungen auf ihre eigenen Modelle haben und genau über die betroffenen Objekte in ihren entsprechenden Modellen berichten.
- **Automatisch lösen**: Automatisch auf eingehende Änderungen in fremden Modellen reagieren und automatisch zugehörige Objekte in den betroffenen Disziplinmodellen ändern.

Heutzutage sind bereits viele CDE-Plattformen in der Lage, Benachrichtigungen für verschiedene Arten von Ereignissen einzurichten, die auf diesen Plattformen auftreten. Solche Ereignisse können das Veröffentlichen neuer Datensätze oder das Melden von Problemen umfassen. Daher kann der einfachste Fall, Benutzer lediglich über neue Modelle zu benachrichtigen, als gelöst betrachtet werden und bedarf keiner weiteren Aufmerksamkeit. Auf der anderen Seite wird eine vollständige Automatisierung kritisch eingestuft und ist ebenfalls nicht beabsichtigt, da sie erheblich in die Urheberschaft der verantwortlichen Ingenieure eingreifen würde. Stattdessen wird ein Ansatz angestrebt, bei dem Benutzer benutzerdefinierte Workflows einrichten können, um weitere eingehende Updates für fremde Modelle zu verarbeiten.

Anschließend kann das behandelte Problem auf die Einrichtung interdisziplinärer Objektbeziehungen konzentriert werden, um anderen Bereichen genau über eingehende

Änderungen und betroffene Komponenten zu berichten. Abb. 5.17 veranschaulicht den Gesamtansatz. Der Verknüpfungsgraph G ergibt sich aus den Disziplingraphen G_A, G_B und G_C sowie den verknüpfenden Kanten α.

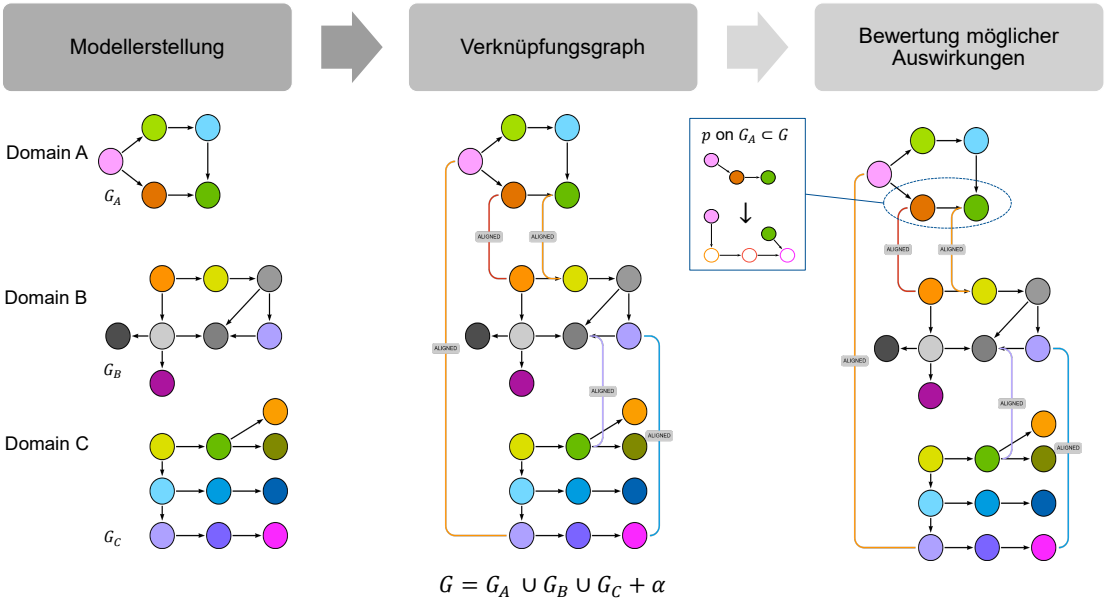


Abbildung 5.17: Disziplinmodelle im Verknüpfungsgraph G mit zusätzlichen Kanten α

Als Voraussetzung stellen alle Bereiche ihre Entwurfsinformationen als Graph bereit. In den meisten Fällen wird erwartet, dass Domänen Informationen als BIM-Modelle nach objektorientierten Paradigmen liefern. Diese Modelle werden dann wie in Abschnitt 5.1 beschrieben versionsüberwacht und inkrementell synchronisiert. Ergänzend werden nun allerdings zusätzliche interdisziplinäre Verweise zwischen Objekten der verschiedenen Disziplinmodelle hergestellt und in den zentralen Graphspeicher integriert. Die zuvor separaten Graphen, die einzelne Disziplinmodelle repräsentiert haben, werden so in einem sogenannten *Verknüpfungsgraph* zusammengeführt, der die Datengrundlage für interdisziplinäre Koordinationsmodelle bildet. Anschließend können eingehende Änderungen in ihrer Auswirkung auf das gesamte (d. h. koordinierte) Modell bewertet werden. Die Auswirkungen einer Transformationsregel p, die auf den Graphen G angewendet wird, der ein bestimmtes Disziplinmodell widerspiegelt, können nun im Rahmen des gesamten Koordinationsmodells (oder des entsprechenden Graphen) bewertet werden.

Die Etablierung von verknüpfenden Beziehungen zwischen Objekten unterschiedlicher Disziplinmodelle ist anspruchsvoll und lässt sich nicht angemessen verallgemeinern. Nichtsdestotrotz können verschiedene räumliche und logische Operatoren dazu beitragen, derartige Beziehungen zu bestimmen. Zur Analyse der Wechselwirkungen zwischen verschiedenen Modellen können beispielsweise die räumlichen Operatoren nach Daum (2018) herangezogen werden. Darüber hinaus existieren verschiedene kommerzielle Ap-

plikationen, die interdisziplinäre Analysemöglichkeiten bereitstellen. Bekannte Werkzeuge sind dabei unter anderem SimpleBIM [1], Desite BIM [2] oder Solibri [3].

Ist die Vernetzung der einzelnen Graphen über Disziplingrenzen hinweg erreicht, können die zusätzlichen Beziehungen genutzt werden, um die Auswirkungen eingehender Commits im Hub zu bewerten und gegebenenfalls individuelle Benachrichtigungen an Projektbeteiligte zu versenden, die sie über neue Aktualisierungen in den möglichen Auswirkungen auf ihre Modelle informieren. Wie in Abschnitt 4.6 dargelegt umfassen die in den Commits enthaltenen Inkremente topologische und semantische Modifikationen. Semantische Modifikationen beeinflussen lediglich Eigenschaften von Knoten ohne topologische Veränderungen im Graphen vorzunehmen. Daher kann das *UniquePath*-Muster, das den bearbeiteten Knoten eindeutig spezifiziert, weitergehend genutzt werden, um Objekte zu identifizieren, die in einer interdisziplinären Beziehung zu dieser Modifikation stehen. Dabei werden alle Knoten genutzt, die im *UniquePath*-Muster spezifiziert sind.

Komplexer gestaltet sich der Umgang mit topologischen Modifikationen. Im besonderen Fokus stehen nun die Kontext- und PushOut-Muster, um die Auswirkungen einer solchen Änderung auf andere Disziplinmodelle zu bewerten. Weitergehend zu unterscheiden ist die Art der topologischen Transformation. Wenn Informationen aus dem Modell entfernt wurden, beginnt das Versionskontrollsystem mit dem Abgleich aller drei Muster (PushOut-, Klebe- und Kontext-Muster). Anschließend werden die Klebe-Kanten gelöscht und das PushOut-Muster aus dem Graphen entfernt. Da interdisziplinäre Objektverknüpfungen mit jedem beliebigen Knoten hergestellt werden können, ist eine zusätzliche Verarbeitung erforderlich. Nach allgemeiner Graphtheorie können in Graphen keine *hängenden* Kanten existieren, bei denen der Start- oder Endknoten entfernt wurden. Rekapituliert man nun nochmal das erläuterte Vorgehen, wie Inkremente auf einen überholten Graph angewendet wurden, wird dieser Zustand dadurch erreicht, dass zuerst alle Klebe-Kanten gelöscht werden. Damit ist der Teilgraph, den das PushOut-Muster spezifiziert, entkoppelt und kann anschließend gelöscht werden. Existieren nun aber neben den im Klebe-Muster spezifizierten Kanten zusätzliche Kanten, die interdisziplinäre Abhängigkeiten zu Objekten anderer Disziplinen beschreiben, müssen diese ebenfalls behandelt werden. Existieren weitere Kanten, die den im PushOut-Muster spezifizierten Teilgraph mit anderen Knoten verbinden als jene, die im Kontext-Muster spezifiziert sind, wird die Anwendung des Inkrements sofort gestoppt, um das Vorhandensein von hängenden Kanten zu verhindern. Weniger kritisch sind Kanten, die interdisziplinäre Abhängigkeiten zu Knoten abbilden, die im Kontext-Muster vorhanden sind. In diesem Fall sollte die Transformation entweder abgelehnt oder die Autoren der zugehörigen Objekte in allen fremden Disziplingraphen sollten informiert werden. Wie zuvor lässt sich dieses Systemverhalten nicht allgemein definieren, sondern sollte im Hinblick auf die Besonderheiten des Projekts festgelegt werden.

[1] https://simplebim.com/ (letzter Zugriff am 30.11.2023)
[2] https://thinkproject.com/products/desite-bim/ (letzter Zugriff am 30.11.2023)
[3] https://www.solibri.com/ (letzter Zugriff am 30.11.2023)

Neben dem Entfernen bestehender Modellkomponenten können Inkremente auch neue Modellkomponenten beziehungsweise entsprechende Teilgraphen in den Graphspeicher hinzufügen und mit dem zugehörigen Disziplinmodell durch das Klebe-Muster verbinden. Dieses Muster spezifiziert allerdings nur jene Kanten, die notwendig sind, um den neu eingefügten Teilgraph korrekt in das entsprechende Disziplinmodell zu integrieren. Beziehungen zu disziplinfremden Bauteilen hingegen sind nicht als Bestandteil des Commits vorgesehen. Natürlich kann aber das Kontextmuster analog zu dem Verfahren des Löschens eines Objekts genutzt werden, um zugehörige Komponenten anderer Modelle zu identifizieren.

Als mögliche Abhilfe wäre denkbar, die in einem Commit übermittelten Klebe-Muster um interdisziplinäre Abhängigkeiten zu erweitern. Deren Ermittlung würde allerdings eine umfangreichere Prozessierung für jeden Nutzer bedeuten, da die bloße Ermittlung des Inkrements für die eigenen Modelle um die Betrachtung interdisziplinärer Abhängigkeiten erweitert werden müsste. Da sich mittlerweile Rollenbilder für die Koordination und das Management föderierter Disziplinmodelle in der modellbasierten Zusammenarbeit etabliert haben, könnte die Ermittlung der verknüpfenden Beziehungen hingegen auch als explizite Aufgabe eines entsprechenden Koordinators vorgesehen werden. Damit wären zwar eingehende Commits, die neue Modellkomponenten in ein Disziplinmodell hinzufügen, erst einmal weiterhin ohne entsprechende interdisziplinären Abhängigkeiten ausgestattet. Die Abstraktion und Verwaltung der einzelnen Versionsinkremente bliebe aber weiterhin auf den Datensatz der individuellen Planung einer Disziplin beschränkt, sodass etwaige Abhängigkeiten von entsprechenden Spezialisten übergreifend und in nachrangiger Art betrachtet werden können. Änderungen in den interdisziplinären Abhängigkeiten können ebenfalls als Inkrement formuliert werden. Relevant wird hierfür insbesondere das durchgehende Beschriften aller Knoten mit dem entsprechenden Zeitstempel sowie der zugehörigen Disziplin, die die Hoheit über einen Disziplinmodell besitzt.

Im folgenden werden nun zwei Szenarien beispielhaft skizziert, um die generelle Anwendbarkeit des vorgestellten Ansatzes zu verdeutlichen. Betrachtet wird ein Geschoss eines Neubaus, bei dem Architektur und Innenplanung kollaborieren. Die Architektur hat ein Disziplinmodell erstellt, das Wände, Fenster und Türen enthält. Im Modell der Innenplanung sind Raumkubaturen sowie diverse Ausstattungsmerkmale wie Tische und Schränke enthalten. Abb. 5.18 zeigt die beiden Disziplinmodelle, das Koordinationsmodell sowie einen Ausschnitt des Verknüpfungsgraphs G.

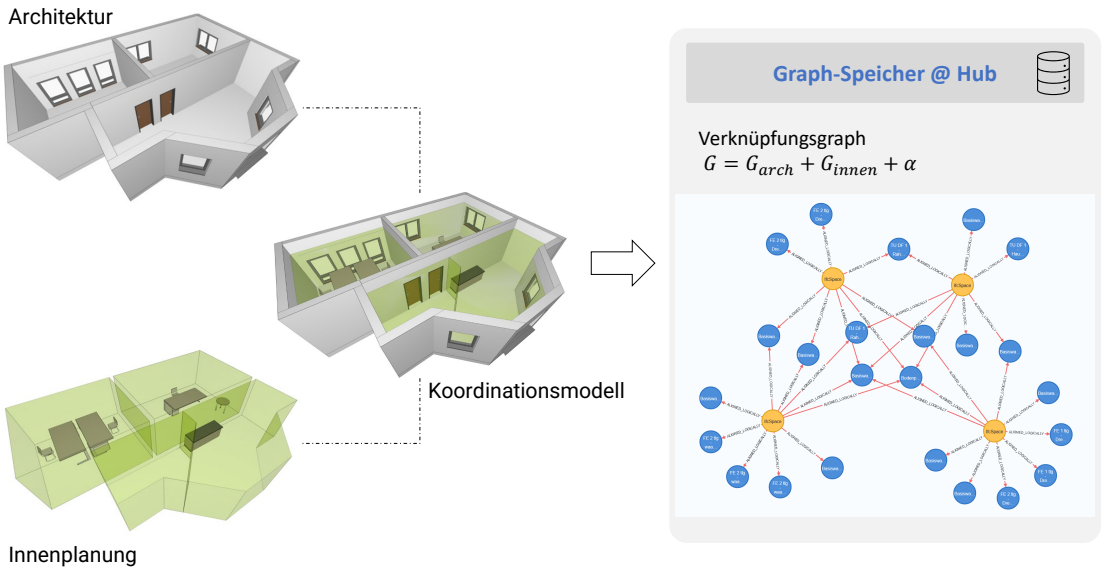


Abbildung 5.18: Verknüpfungsgraph für ein Koordinationsmodell, das aus den Disziplinen Architektur und Innenplanung entsteht

Nun werden zwei Änderungen betrachtet, die jeweils von der Architektur übermittelt und an den Hub gepusht werden. Abb. 5.19 illustriert das erste Änderungsszenario, bei dem eine Innenwand des Architekturmodells verschoben wird. Es handelt sich dabei um eine rein semantische Modifikation, die gemäß dem in Abschnitt 4.6.1 dargelegten Datenmodells beschrieben wird. Da zwischen der Innenwand und den angrenzenden Räumen entsprechende Abhängigkeiten im Verknüpfungsgraph modelliert wurden, kann durch Analyse des im Inkrement gegebenen *UniquePath*-Musters direkt die Abhängigkeit zu den Raumgeometrien von *Office 1* und *Office 2* explizit ermittelt werden. Im illustrierten Fall wäre es darüber hinaus denkbar, auf Basis der Transformationsattribute *ValueInit* und *ValueUpdt* einen Verschiebungsvektor zu ermitteln, der anschließend auf die geometrischen Repräsentationen beider Räume angewendet werden könnte. Da deren exakte Modellierung allerdings spezifisch von den eingesetzten Autorenwerkzeugen abhängt, kann keine pauschale Aussage hinsichtlich der exakten notwendigen Transformation dieser Geometrie getroffen werden. Vielmehr obliegt es der zugehörigen Disziplin, bei entsprechender Benachrichtigung über Änderungen in der Architektur notwendige Maßnahmen für die Räume im Modell der Innenplanung zu ergreifen.

Im zweiten Beispiel wird das Architekturmodell um eine neue Innenwand erweitert. Das zugehörige Inkrement beinhaltet die bekannten Muster. Das PushOut-Muster spezifiziert jenen Teilgraph, der die neue Wand inklusive ihrer Geometrie und zugehöriger Eigenschaften repräsentiert. Das Klebemuster verbindet das PushOut-Muster mit den zugehörigen Knoten des Kontext-Musters. Das Inkrement ist in Abb. 5.20 qualitativ dargestellt.

Die gestrichelt dargestellten Beziehungen, die notwendig wären, um den Verknüpfungsgraph entsprechend um die neu entstandenen Abhängigkeiten zwischen den veränderten Architekturelementen und der Innenplanung zu erweitern, sind nicht im Inkrement

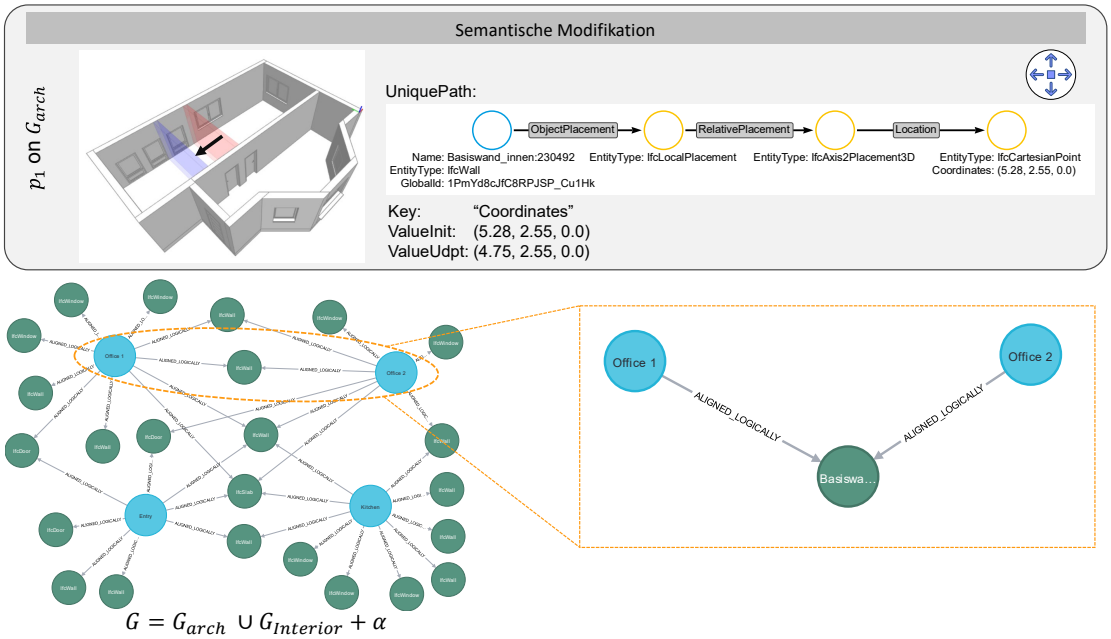


Abbildung 5.19: Nutzung eines Verknüpfungsgraphs: Verschieben einer bestehenden Wand, die Abhängigkeiten zu den Raumgeometrien des Innenplaners aufweist

enthalten. Diese müssten wie beschrieben entweder durch einen entsprechenden BIM-Koordinator in einem weiteren Patch hinzugefügt werden oder entsprechend als Erweiterung des Inkrements durch die Archtitektur an den zentralen Hub gesendet werden. Hierzu können die in Abschnitt 5.1 eingangs erwähnten Labels entsprechend genutzt werden, um die Graphen einzelner Disziplinen einerseits ihrer Bezeichnung und andererseits basierend auf der zugehörigen Versionsnummer anzusprechen.

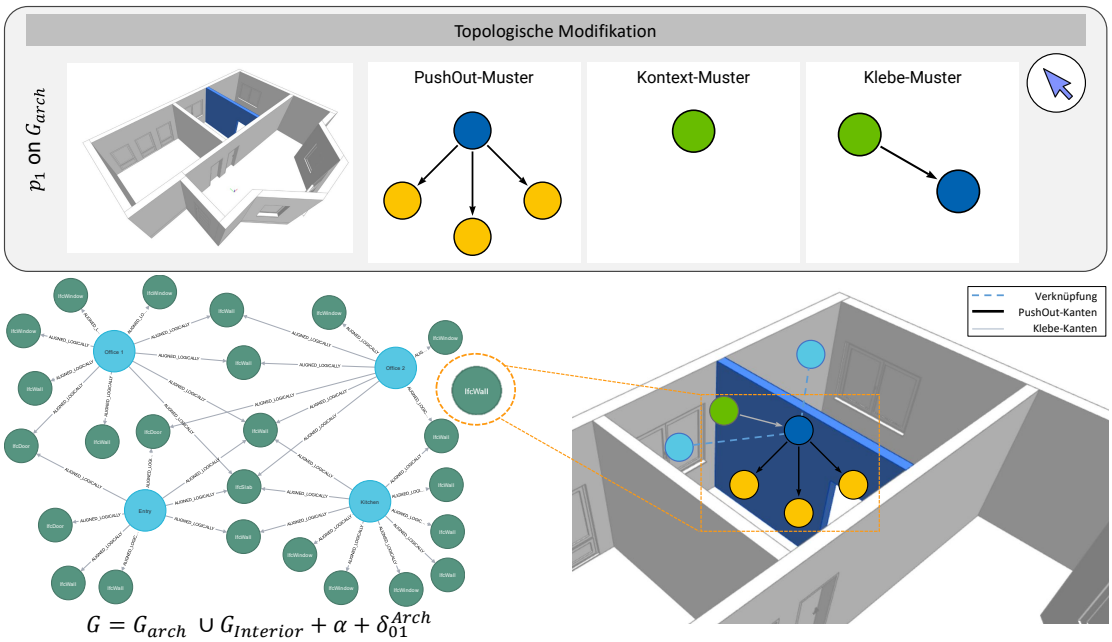


Abbildung 5.20: Nutzung eines Verknüpfungsgraphs: Einfügen einer neuen Innenwand

5.5 Einordnung und Zusammenfassung

Dieses Kapitel hat die zuvor entwickelte Methodik zur Erfassung, Übertragung und Integration von Modellinkrementen in den Kontext eines vollwertigen Versionskontrollsystems gesetzt. Neben dem Versionsmanagement auf einem einzelnen Client mit den Befehlen *add* und *commit* ist insbesondere die Einführung des zentralen Hubs mit den zugehörigen Befehlen *push* und *pull* zur Synchronisierung ein wesentlicher Schritt hin zur zeitlich flexiblen Synchronisation von Modelländerungen zwischen teilnehmenden Projektakteuren. Die zeitlichen Intervalle zum Austausch inkrementeller Änderungen können dabei vollkommen flexibel durch die Nutzer gewählt werden und so exakt an die Bedürfnisse und Randbedingungen der jeweiligen Projekte angepasst werden. Kurze Sychronisationszyklen ermöglichen dabei die interdisziplinäre Zusammenarbeit nahe eines Echtzeitsystems, während gleichzeitig auch explizite Übertragungszeitpunkte im Tages- oder Wochenrythmus angewandt werden können. Darüber hinaus kann jeder Nutzer in seinem lokalen Repository beliebig viele Versionsschritte verwalten und erst nach Fertigstellung aller notwendigen Abwägungen die Daten übertragen.

Neben den vorgestellten Überlegungen zur chronologischen Übertragung wurden weitere Aspekte zur parallelen Entwicklung von Versionszweigen und deren späterer Integration erörtert. Durch die entsprechende Analyse der semantischen und topologischen Transformationen von Inkrementen verschiedener Branches können aussagekräftige Vorhersagen getroffen werden, inwieweit divergierende Modellzustände wieder in einem Modell vereint werden können. Außerdem können Probleme erkannt werden, die entweder durch eine einfache Nutzerentscheidung gelöst werden können oder eine umfangreichere Betrachtungen notwendig machen. Darüber hinaus wurde in Abschnitt 5.4 dargelegt, welcher zusätzliche Nutzen entsteht, wenn interdisziplinäre Abhängigkeiten im Graphspeicher des Kollaborations-Hubs modelliert werden und inwiefern diese für die präzise Benachrichtigung anderer Disziplinen relevant sein können.

Die gezeigten Beispiele einer solchen Verknüpfung von Modellkomponenten im zentralen Hub verdeutlichen, dass in einfachen Fällen eine weitergehende Automatisierung von Reaktionen auf disziplinfremde Commits denkbar ist. Insbesondere in Situationen, in denen die Auswirkung disziplinfremder Änderungen eindeutig auswertbar sind und solche Änderungen mehrfach in einem Planungsprozess auftreten können, kann eine entsprechende Automatisierung Sinn machen. Diese ist insbesondere für Abhängigkeiten in der Platzierung von Objekten zu erwarten (beispielsweise bei der Modifikation von Wand- und Stützenrastern). Komplexere Modifikationen mit verschiedenen semantischen und topologischen Transformationen stellen hierbei allerdings eine anspruchsvolle Herausforderung für vollständige Automationen dar. Hier kann dennoch ein großer Mehrwert darin liegen, dass Disziplinen ausgehend von Änderungen disziplinfremder, aber abhängiger Modellkomponenten sensitiv informiert werden können.

Kapitel 6

Anwendungsbeispiel

Um die erläuterten Ansätze in einem realen Kontext zu demonstrieren, präsentiert das folgende Kapitel eine umfangreiche Fallstudie. Dabei werden Modelle der Disziplinen Architektur und Tragwerksplanung betrachtet.

Als Beispiel dient ein Modell eines fiktiven Bauvorhabens, bei dem ein Ausstellungspavillon entworfen wird. Das Architekturmodell ist im Ausgangszustand in Abb. 6.1 dargestellt. Dieses wurde mit dem BIM-Autorenwerkzeug Autodesk Revit 2024 [1] erstellt.

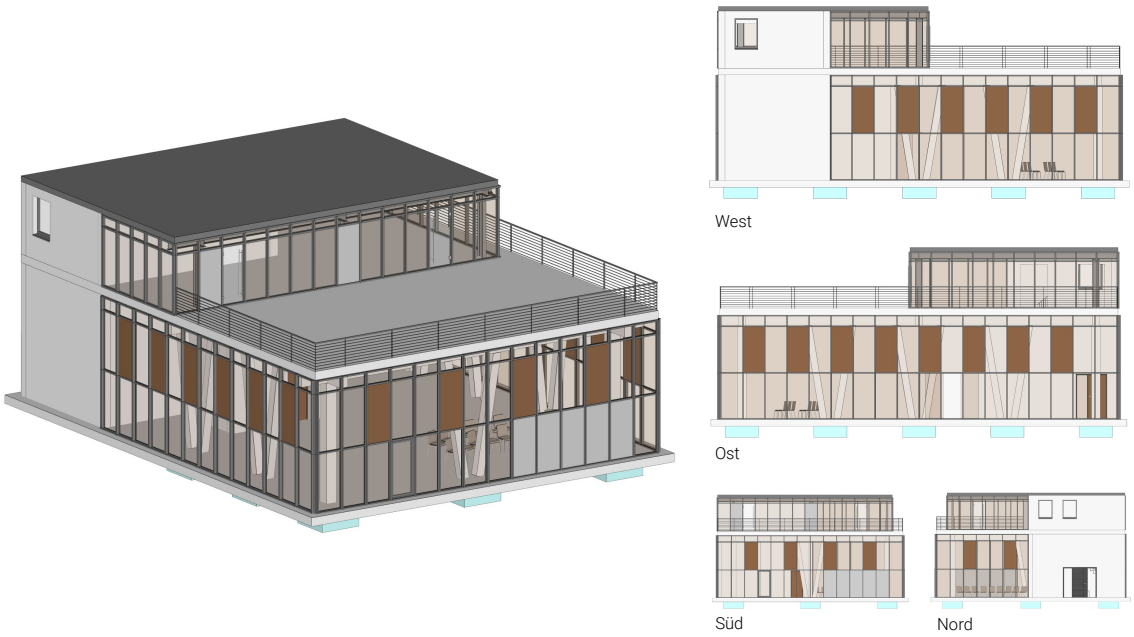


Abbildung 6.1: Betrachtetes Architekturmodell

Für die Tragwerksplanung wurde ausgehend von den im Architekturmodell enthaltenen Elementen ein eigenes Disziplinmodell ausgearbeitet. Dieses ist in Abb. 6.2 dargestellt und wurde mithilfe Tekla Structures 2023 des Herstellers Trimble [2] erstellt. Als initiale Grundlage diente hierfür das Architekturmodell. Beide Modelle wurden nach der Modellierung als IFC-Modell exportiert. Das Tragverhalten des Gebäudes ergibt sich im wesentlichen durch Träger in den Hauptrichtungen des Gebäudes, die auf lastabtragenden Stützen in den Fassadenbereichen vertikale Lasten der Deckenplatten abtragen. Zur Aussteifung dient ein Treppenhaus im Nordosten, welches für die Abtragung horizontaler Lasten angesetzt wird. Im Architekturmodell sind zusätzliche Wände aus Mauerwerk vorgesehen, die der

[1] https://www.autodesk.de/products/revit/ (letzter Zugriff am 14.02.2024)
[2] https://www.tekla.com/de/produkte/tekla-structures (letzter Zugriff am 14.02.2024)

Abtrennung von Sanitäranlagen dienen. Diese werden allerdings nicht als lastabtragend angesetzt und werden folglich nicht im Tragwerksmodell abgebildet.

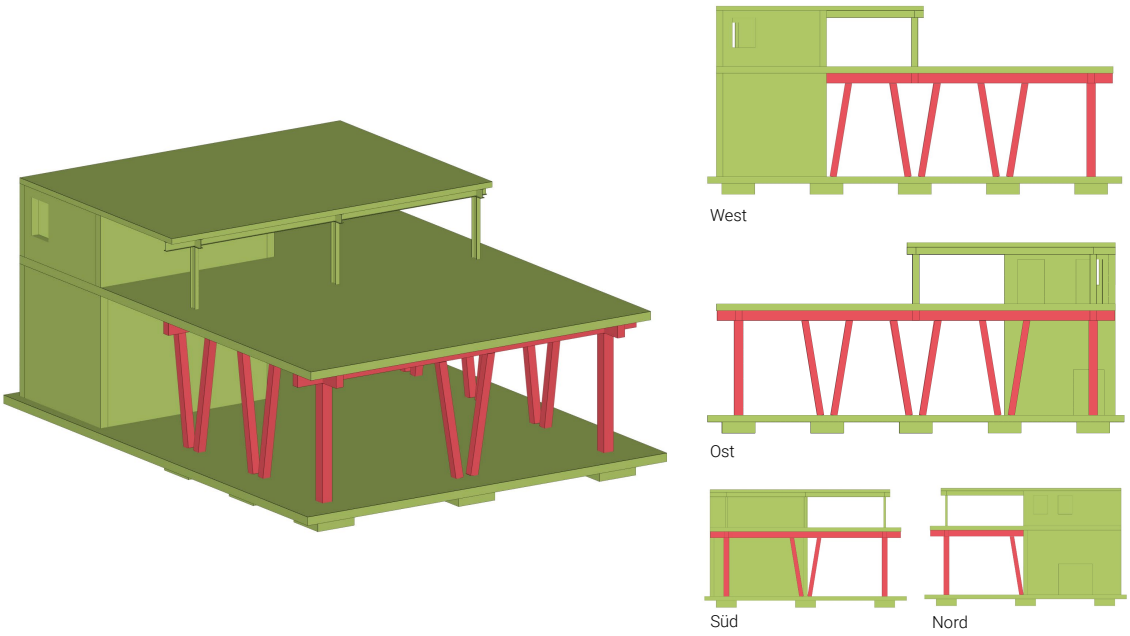


Abbildung 6.2: Betrachtetes Tragwerksmodell

Der aus dem Architekturmodell resultierende Graph besteht aus 30999 Knoten und 107262 gerichteten Kanten. Das Tragwerksmodell wird in einem Graph mit 1114 Knoten sowie 3840 Kanten repräsentiert. Die Ursache für die im Vergleich zum Architekturmodell deutlich kompaktere Form des Tragwerksmodells sind die gewählten geometrischen Repräsentationen sowie die erheblich geringere Anzahl an Bauteilen, die im IFC-Modell enthalten sind. Da das Fassadensystem aus zahlreichen Einzelkomponenten besteht, schlägt sich dessen Abbildung im Graph mit besonders vielen Knoten und Kanten nieder.

Leider verfügte das verwendete Autorensystem Tekla Structures zur Erstellung des Tragwerksmodells nicht über die Möglichkeit, gleichbleibende Identifikationsmerkmale über mehrere Exportzyklen zu verwenden. Daher wurde in diesem Fall eine spezifische Nachbearbeitung der exportierten Tragwerksmodelle implementiert, die die *GlobalId*-Attribute aller Primärobjekte basierend auf dem unveränderten Objektnamen anpasst und damit persistent über alle betrachteten Versionen des Tragwerksmodells macht.

6.1 Verbindungsgraph und Koordinationsmodell

Nach der Erstellung beider Disziplinmodelle und deren Bereitstellung im Hub werden die interdisziplinären Beziehungen zwischen den Komponenten des Architektur- und des Tragwerksmodells ermittelt. In frühen Projektphasen sind in den Modellen noch keine

Räume oder sonstige Ausrüstungskomponenten modelliert. Daher erfolgt die Modellierung des Verknüpfungsgraphs vor allem durch die folgenden Kriterien:

- Räumliche Strukturierung in Gebäude und Geschosse
- Abhängigkeiten von Komponenten aufgrund ihrer Platzierung im dreidimensionalen Raum

Die letztgenannte Abhängigkeit wird durch die Analyse der geometrischen Bauteilformen und deren Überlappung ermittelt. Abb. 6.3 zeigt die Komponenten des Tragwerksmodells, zu denen ein entsprechendes Gegenstück im Architekturmodell ermittelt werden konnte. Wie erwartet konnten für alle tragenden Bauteile des Tragwerksmodells passende Äquivalente im Architekturmodell identifiziert werden. Die gewählte Färbung gruppiert Bauteile, die denselben Bauteiltyp verwenden.

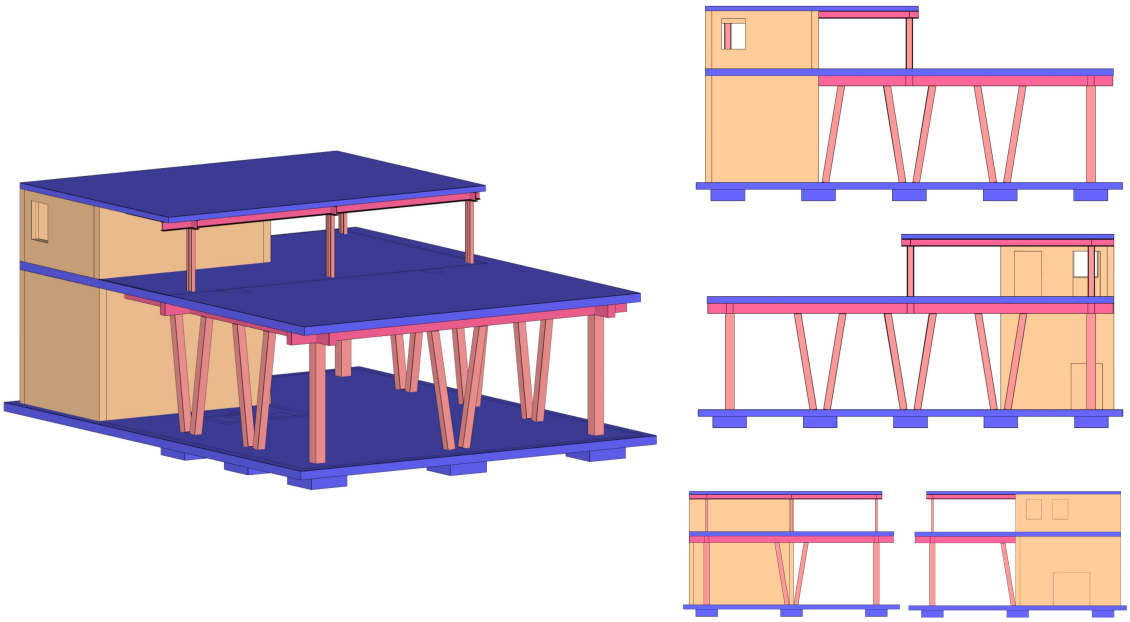


Abbildung 6.3: Objekte des Tragwerksmodells, zu denen passende Gegenstücke im Architekturmodell identifiziert werden konnten

6.2 Untersuchte Prozesse

Untersucht wird nun der in Abb. 6.4 dargestellte Prozess. Für die Bezeichnung des Architekturmodells wird abkürzend der Term *ARC-Modell* verwendet, der auch in der Bezeichnung der zugehörigen Graphen genutzt wird. In gleichem Sinne bezeichnet *TW-Modell* das Tragwerksmodell und $G_{i,TW}$ den zu einer spezifischen Version zugehörigen Graph, der aus dem Tragwerksmodell generiert wurde.

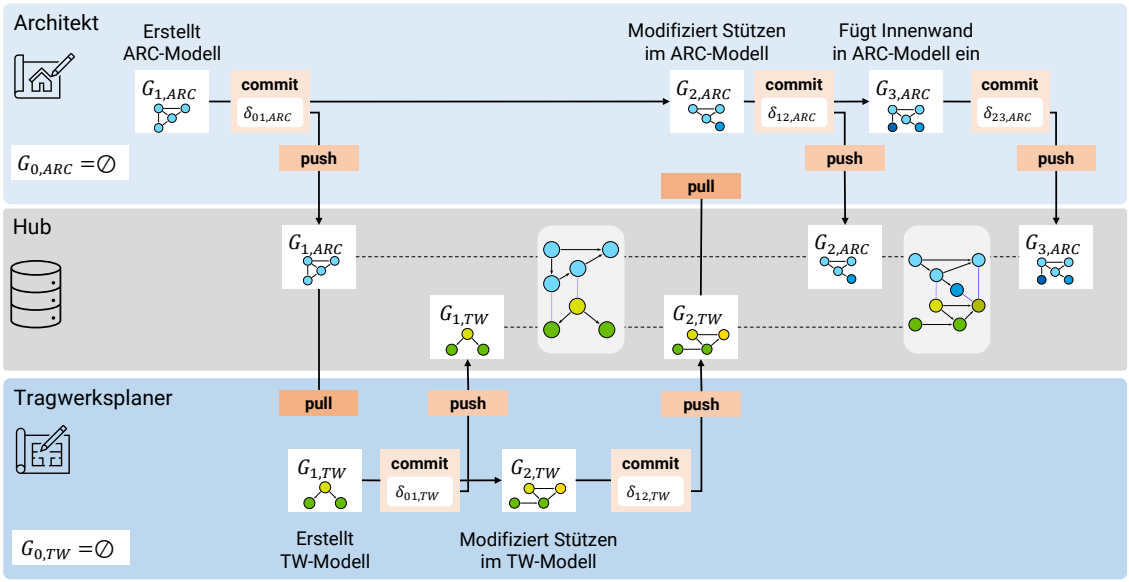


Abbildung 6.4: Kollaborativer Prozess zwischen Architekt und Tragwerksplaner

Der erste Commit δ_{01} übermittelt die erste Version des Modells. Analog zu Git hat der erste Commit keine Vorgängerversion, auf den dieser referenzieren könnte. Damit das in Abschnitt 4.5 erläuterte Diff-Verfahren zur Ermittlung der unveränderten Teile des Graphs dennoch gemäß der formal aufgestellten Kriterien funktioniert, wird als initiale Version G_{init} ein leerer Graph angenommen. Durch den Abgleich des leeren Graphs G_0 mit G_1 sind beide Eingabeargumente der Diff-Funktion gegeben. Das zugehörige Inkrement δ_{01} wird folglich passend ermittelt. Es erzeugt alle Knoten und Kanten, die im Graph G_1 enthalten sind. In allen folgenden Prozessschritten wird jeweils die Vorgängerversion des Graphs für die Diff-Operation herangezogen und die ermittelten Modifikationen in das entsprechende Inkrement δ kodiert.

6.2.1 Modifikation der Stützen im Tragwerksmodell

Der Tragwerksplaner hat zu Beginn die Stützenquerschnitte aus dem Architekturmodell übernommen und auf dieser Basis das initiale Tragwerksmodell erstellt. Betrachtet wurden dabei alle für das Tragverhalten des Bauwerks relevanten Bauteile. Konkret wurden sämtliche Stützen, Balken, Fundamente und Decken sowie die Wände des Treppenhauses, die als aussteifender Kern wirken, aus dem Architekturmodell importiert. Die Windlasten, die auf die Außenfassade wirken, wurden ersatzweise auf die Deckenplatten als linienförmige Last angesetzt, sodass die individuellen Fassadenelemente nicht Teil des Tragwerksmodells sind. An den Stellen der Türen und Fenster wurden entsprechende Durchbrüche und Aussparungen im Tragswerksmodell berücksichtigt, die mit der IFC-Entität `IfcVoidingFeature` modelliert wurden.

Als Resultat der Tragwerksbemessung müssen die drei Stützen in den Gebäudeecken in ihrem Querschitt von $30x30cm^2$ auf $40x40cm^2$ vergrößert werden. In Abb. 6.5 ist die Änderung visualisiert, die das Inkrement $\delta_{01,TW}$ hervorruft.

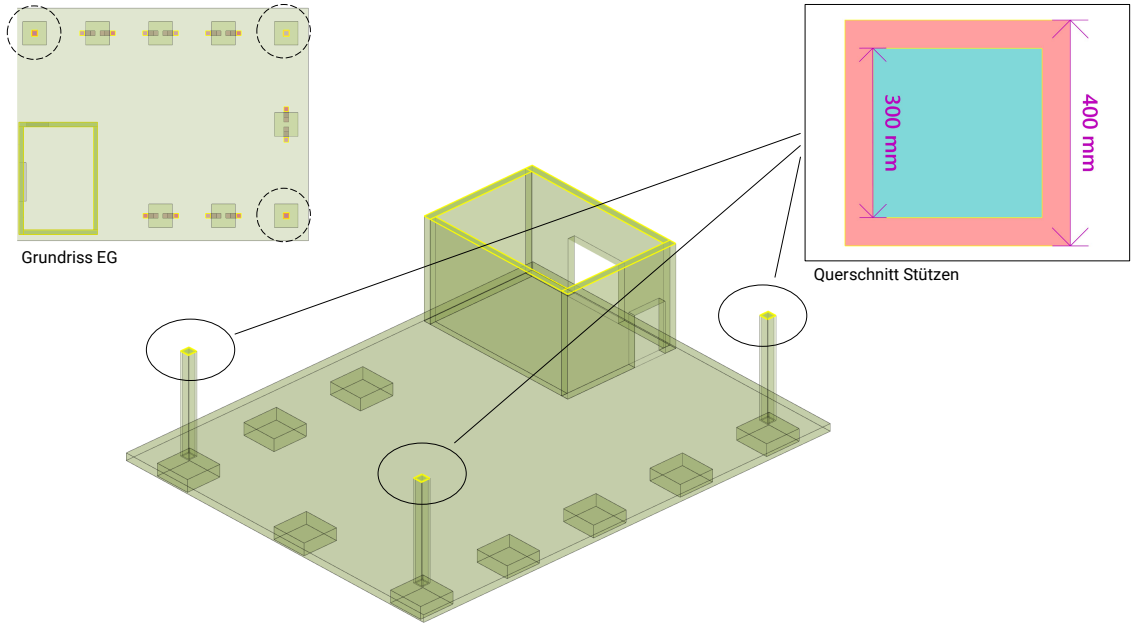


Abbildung 6.5: Visualisierung der Modifikationen, die das Inkrement $\delta_{12,TW}$ bewirkt

Tabelle 6.1 beschreibt alle semantischen Modifikationen, die im Inkrement $\delta_{12,TW}$ enthalten sind.

Node	Key	ValueInit	ValueUpdt
94	Description	300*300	400*400
94	Name	STB Stütze - rechteckig: STB 300 x 300:2520640	STB Stütze - rechteckig: STB 400 x 400:2520640
43	ProfileName	300*300	400*400
41	CoordList	((-150.0, -150.0), (150.0, -150.0), (150.0, 150.0), (-150.0, 150.0))	((-200.0, -200.0), (200.0, -200.0), (200.0, 200.0), (-200.0, 200.0))
86	Description	300*300	400*400
86	Name	STB Stütze - rechteckig: STB 300 x 300:2521453	STB Stütze - rechteckig: STB 400 x 400:2521453
61	Description	300*300	400*400
61	Name	STB Stütze - rechteckig: STB 300 x 300:2521492	STB Stütze - rechteckig: STB 400 x 400:2521492

Tabelle 6.1: Semantische Modifikationen im Inkrement $\delta_{12,TW}$

Die Lage der zu modifizierenden Knoten ergibt sich durch die Nutzung eindeutiger Pfade. Für eine bessere Übersichtlichkeit wurden sämtliche Pfade in ein gesamtheitliches Muster kombiniert, welches in Abb. 6.6 dargestellt ist. Die Knotennummerierung ist dabei willkürlich gewählt. Wie beschrieben müssen Sekundärknoten immer durch die Konstruktion eines Pfads identifiziert werden, der mindestens einen Primärknoten mit eindeutigem Merkmal beinhaltet.

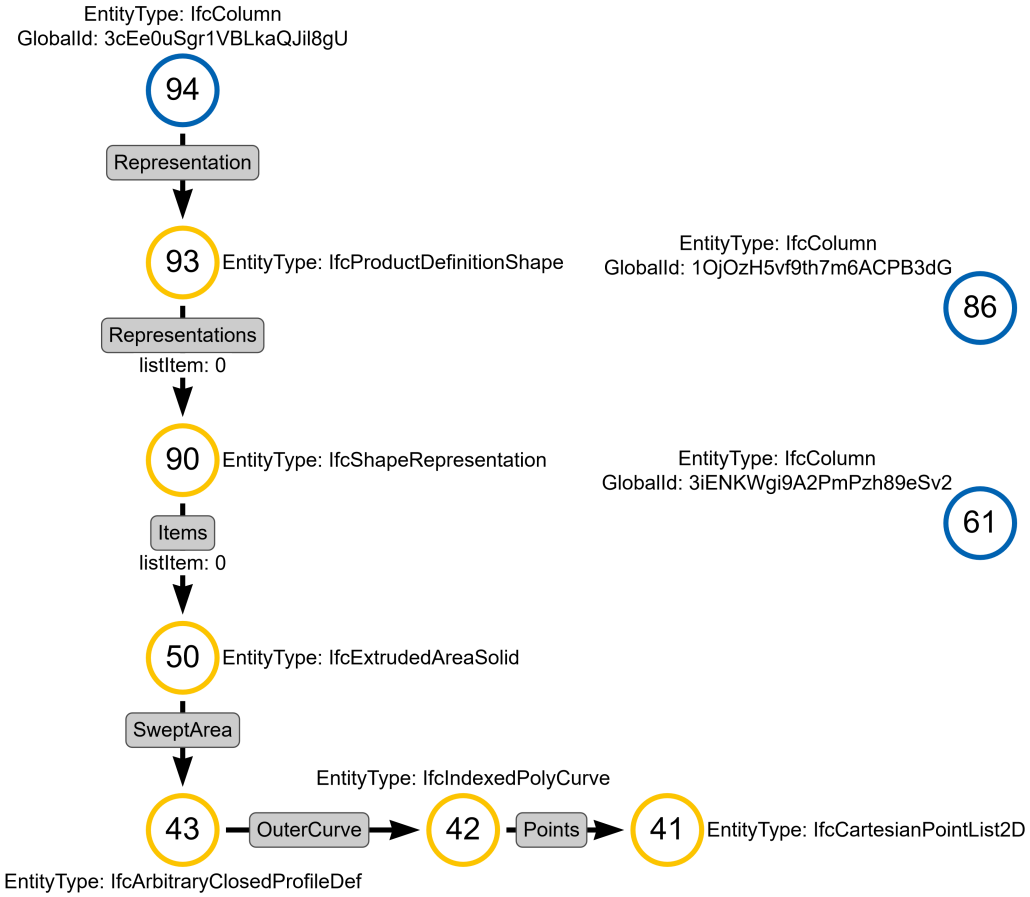


Abbildung 6.6: Muster zur Ermittlung der Knoten mit semantischen Modifikationen, um das Inkrement $\delta_{12,TW}$ anzuwenden

Geändert wurden für jede Stütze die Attribute *Name* und *Description* im jeweiligen Primärknoten, der eine Instanz von `IfcColumn` abbildet. Darüber hinaus wurde das Attribut *ProfileName* im Sekundärknoten 43 geändert, der eine Instanz der IFC-Entität `IfcArbitraryClosedProfileDef` repräsentiert. Die Modifikation der eigentlichen Querschnittsgeometrie erfolgt schließlich im Sekundärknoten 41, der als Instanz der Entität `IfcCartesianPointList2D` den Stützenquerschnitt als Polygonzug beschreibt.

Insgesamt enthält das Inkrement $\delta_{12,TW}$ acht semantische Modifikationen. Zu den zuvor erläuterten Modifikationen kommen die Änderungen an den Primärknoten für die Stützen mit GlobalId *1OjOzH5vf9th7m6ACPB3dG* und *3iENKWgi9A2PmPzh89eSv2* hinzu. Zu erwähnen sei, dass alle drei Stützen die gleiche geometrische Repräsentation nutzen. Abb. 6.7 zeigt den Teilgraph mit jenen Knoten, die durch die semantischen Modifikationen verändert werden. Daraus wird ersichtlich, warum die Modifikation zur Veränderung

des Querschnitts nur einmal gegeben ist. Alle Stützen referenzieren auf die gleiche Extrusionsgeometrie. Knoten, deren Attribute modifiziert werden, sind zusätzlich mit einem x gekennzeichnet.

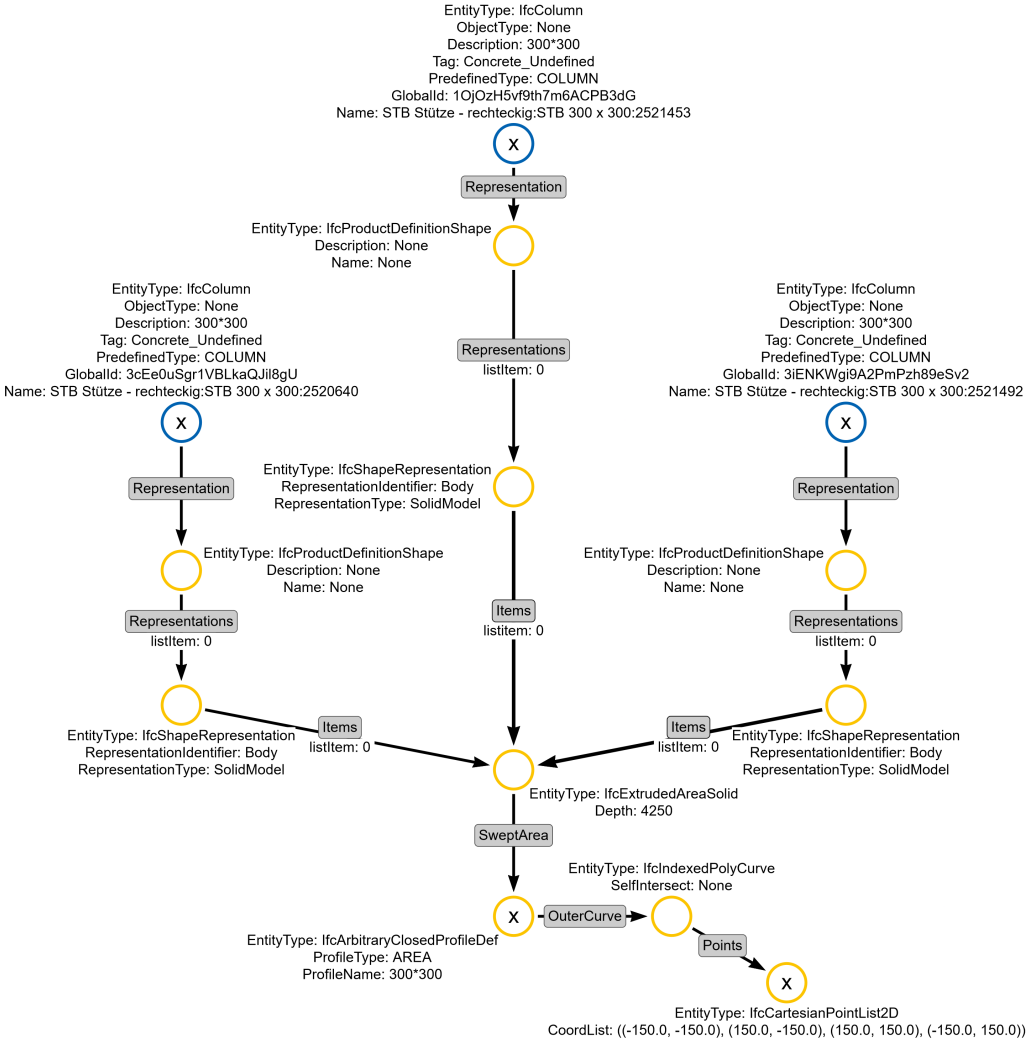


Abbildung 6.7: Auszug des Graphs, der das Tragwerksmodell in der Version 1 repräsentiert

Die notwendigen Cypher-Befehle zur Ausführung aller acht semantischen Modifikationen sind in Anhang A.1.1 gegeben. Dabei können entweder alle relevanten Knoten nach einander mit einem eindeutigen Pfad gesucht und das veränderte Attribut angepasst werden oder zuerst alle Knoten mit dem aggregierten Muster ermittelt und dann sämtliche Änderungen auf einmal angewendet werden. Beide Varianten sind in Anhang A.1.1 dargestellt.

6.2.2 Modifikation der Stützen im Architekturmodell

Nach der Änderung im Tragwerksmodell wird das Inkrement $\delta_{12,TW}$ allen Projektbeteiligten über den Kollaborations-Hub zur Verfügung gestellt. Der Architekt synchronisiert die Inkremente mit jenen auf dem Hub und erhält so die aktuelle Version des Tragwerksmodell. Diese hält er als lokale Kopie in seinem lokalen Repository vor und analysiert die vom Trag-

werksplaner vorgenommenen Änderungen. Der Architekt bewertet die disziplinfremden Auswirkungen und erarbeitet anschließend eine neue Version, um nach der Änderung des Tragwerksmodells wiederum eine interdisziplinäre Konsistenz aller Disziplinmodelle zu erreichen. In der Folge passt er die Stützenquerschnitte in seinem Autorensystem an und exportiert eine neue Modellversion des Architekturmodells. Anschließend wird der Commit $\delta_{12,ARC}$ erstellt, der diese Modifikation enthält. Die Auswirkungen auf die Modellgeometrie sind in Abb. 6.8 dargestellt.

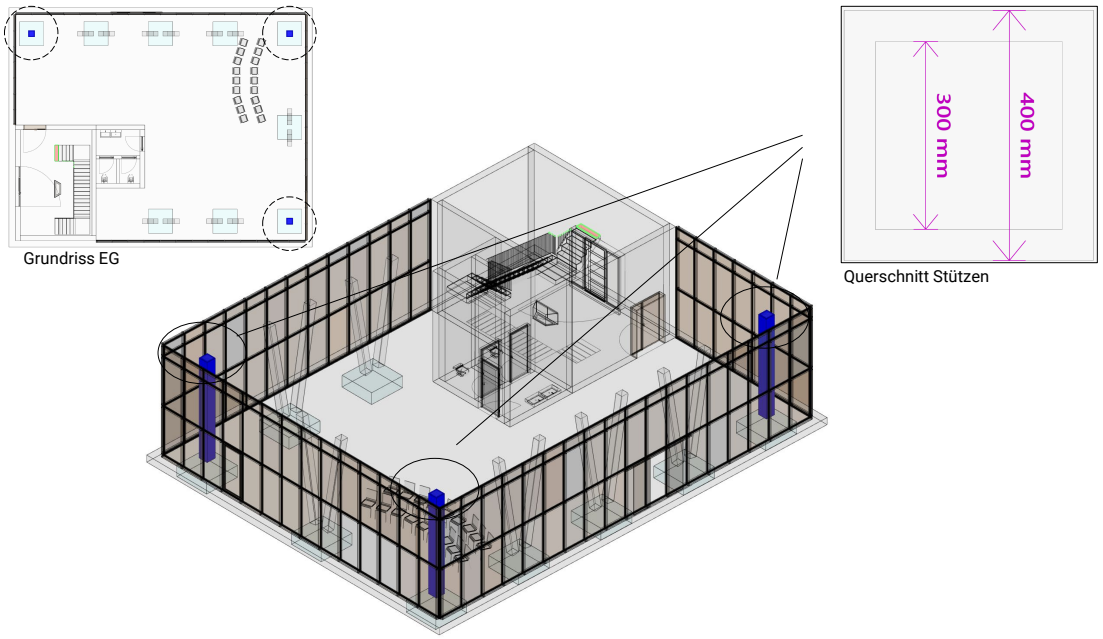


Abbildung 6.8: Änderung der Stützen im Architekturmodell

Wie zuvor im Tragwerksmodell handelt es sich bei der Anpassung der Stützenquerschnitte auch hier um eine rein semantische Modifikation. Allerdings sind im Architekturmodell deutlich mehr Knoten betroffen als im Tragwerksmodell. Abb. 6.9 zeigt alle Pfade, die für das eindeutige Auffinden der zu modifizierenden Knoten notwendig sind, in einem kombinierten Muster.

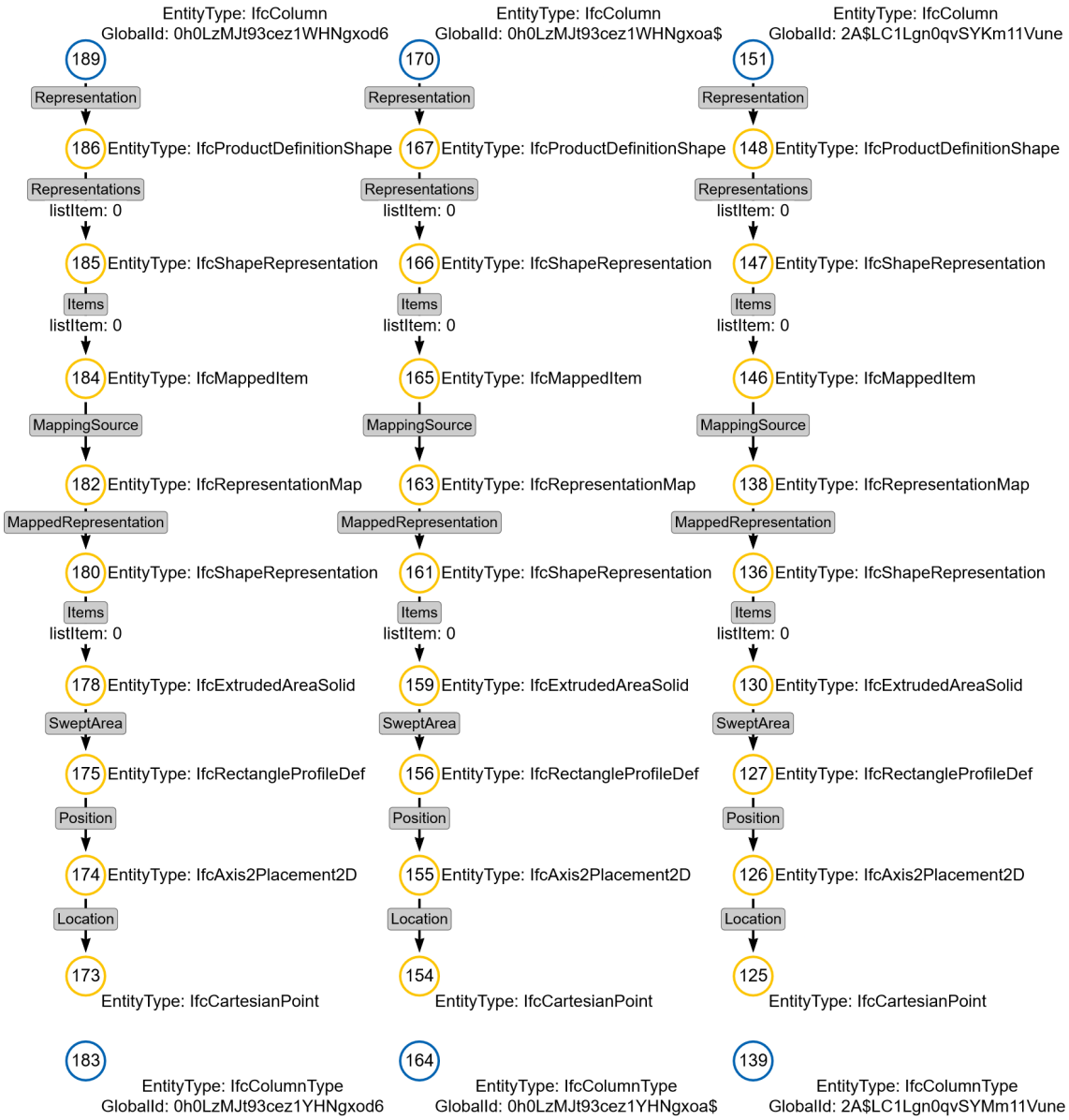


Abbildung 6.9: Muster zur Ermittlung der Knoten mit semantischen Modifikationen, um das Inkrement $\delta_{12,ARC}$ anzuwenden

Tabelle 6.2 beinhaltet alle semantischen Modifikationen, die im Inkrement $\delta_{12,ARC}$ enthalten sind. Deren Anwendung auf eine veraltete Version des Graphs gestaltet sich analog zu den vorherigen Ausführungen und ist in Anhang A.1.2 in Cypher-Sprache dargestellt. Insgesamt werden mit diesem Inkrement 12 Knoten und 24 Attribute modifiziert.

Node	Key	ValueInit	ValueUpdt
189	ObjectType	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400
189	Name	STB Stütze - rechteckig: STB 300 x 300:2521492	STB Stütze - rechteckig: STB 400 x 400:2521492
175	YDim	0.3000000000000011	0.4
175	XDim	0.29999999999999893	0.4
175	ProfileName	STB 300 x 300	STB 400 x 400
173	Coordinates	(-1.0824674490095276e-15, 0.0)	(0.0, 2.1649348980190553e-15)
170	ObjectType	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400
170	Name	STB Stütze - rechteckig: STB 300 x 300:2521453	STB Stütze - rechteckig: STB 400 x 400:2521453
156	YDim	0.30000000000000004	0.4
156	XDim	0.29999999999999893	0.4
156	ProfileName	STB 300 x 300	STB 400 x 400
154	Coordinates	(-1.082467449005276e-15, 0.0)	(0.0, 0.0)
151	ObjectType	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400
151	Name	STB Stütze - rechteckig: STB 300 x 300:2520640	STB Stütze - rechteckig: STB 400 x 400:2520640
127	YDim	0.30000000000000004	0.4
127	XDim	0.30000000000000004	0.4
127	ProfileName	STB 300 x 300	STB 400 x 400
125	Coordinates	(5.412337245047638e-16, 0.0)	(0.0, 0.0)
183	Tag	2077222	2077224
183	Name	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400
164	Tag	2077222	2077224
164	Name	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400
139	Tag	2077222	2077224
139	Name	STB Stütze - rechteckig: STB 300 x 300	STB Stütze - rechteckig: STB 400 x 400

Tabelle 6.2: Semantische Modifikationen im Inkrement $\delta_{12,ARC}$

Die im Inkrement $\delta_{12,ARC}$ enthaltenen Modifikationen in den Knoten 175 und 156 weisen Parallelen zu den Änderungen auf, die zuvor bereits für die Knoten 43 und 41 im Inkrement $\delta_{12,TW}$ für das Tragwerksmodell beschrieben wurden. Während für die erstgenannten Knoten Attributname und Wertmodifikation übereinstimmen, ist die Parallele im letztgenannten Fall durch weitergehende Interpretation der Werte ableitbar. Sofern eine derartige Modifikation häufiger in einem kollaborativen Prozess vorgenommen wird, wäre es denkbar, nachgelagerte Prozessierungsroutinen vorzuhalten, die eingehende Commits aus anderen Disziplinen weitergehend aufbereiten und nach der Freigabe des Nutzers automatisiert auf das Disziplinmodell des Architekten anwenden.

Bei genauerer Betrachtung der modifizierten Knoten sowie der geänderten Attributwerte ist zu erkennen, dass einzelne Änderungen eher beiläufigen Charakter haben. Die Änderungen an den Knoten 173, 154 und 125 weisen eine unterschiedliche Wert-Darstellung auf, aber unterscheiden sich in ihrer Bedeutung nur marginal. Positiv ist zu bewerten, dass auch solche minimalen Änderungen richtig detektiert werden. Treten solche marginalen Abweichungen allerdings häufiger auf, kann die Größe des abgeleiteten Inkrements schnell steigen, ohne dass ein erheblicher Nutzen dieser übertragenen Änderungen zu erwarten wäre.

6.2.3 Hinzufügen einer neuen Wand in das Architekturmodell

Später wird eine weitere Änderung im Architekturmodell vorgenommen. In dieser Anpassung wird eine neue Wand eingefügt, um einen Bereich für Vorträge von der übrigen Ausstellungsfläche abzugrenzen. Die Änderung zeigt Abb. 6.10.

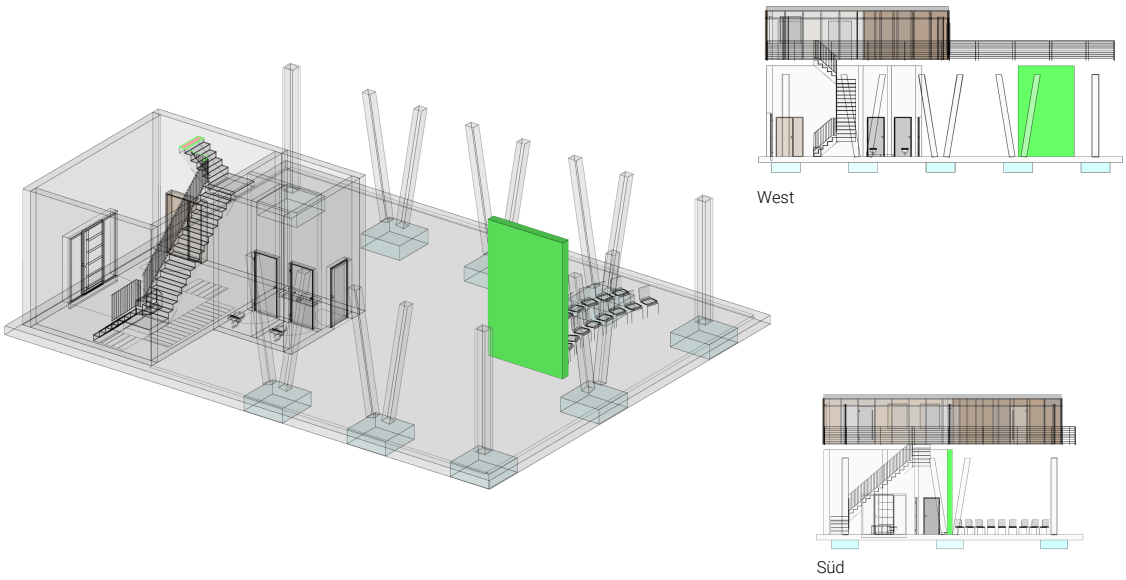


Abbildung 6.10: Einfügen einer neuen Innenwand im Architekturmodell

Das Einfügen einer neuen Modellkomponente geht mit Änderungen in dem topologischen Gefüge des Graphs einher und wird mit dem Inkrement $\delta_{23,ARC}$ erfasst. Diese Modifikation wird durch durch die PushOut-, Klebe- und Kontext-Muster beschrieben. In Abb. 6.11 ist das Kontext-Muster illustriert. Abb. 6.12 zeigt den neu einzufügenden Graph, der durch das PushOut-Muster spezifiziert wird. Das Muster folgt im wesentlichen den Strukturen, die auch schon in Abschnitt 4.9.2 illustriert wurden. Etwas umfangreicher gestaltet sich jener Teil des Graphs, der die geometrische Repräsentation der Wand abbildet. Zur besseren Übersicht wurden alle Knoten der geometrischen Repräsentation in hellgrün und zur Platzierung in magenta eingefärbt. Zudem wurden alle Dezimalzahlen auf zwei Nachkommastellen gerundet, um die Übersichtlichkeit in der Darstellung zu erhöhen. In Abb. 6.13 sind die verbindenden Kanten des Klebe-Musters dargestellt.

Insgesamt werden mit dieser Transaktion 41 neue Knoten sowie $42 + 14 = 56$ neue Kanten hinzugefügt, die entsprechend im PushOut- und im Klebe-Muster definiert sind. Das Kontext-Muster besteht aus 19 Knoten sowie 15 Kanten.

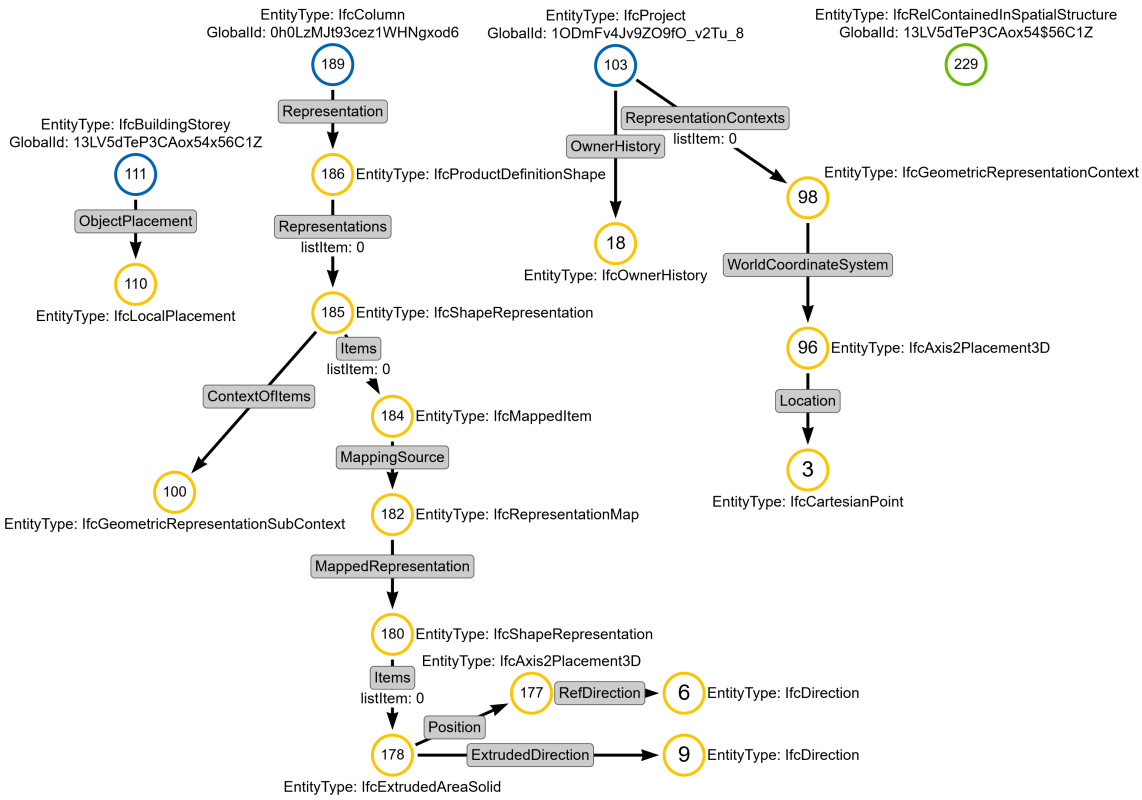


Abbildung 6.11: Kontext-Muster des Inkrements $\delta_{23,ARC}$

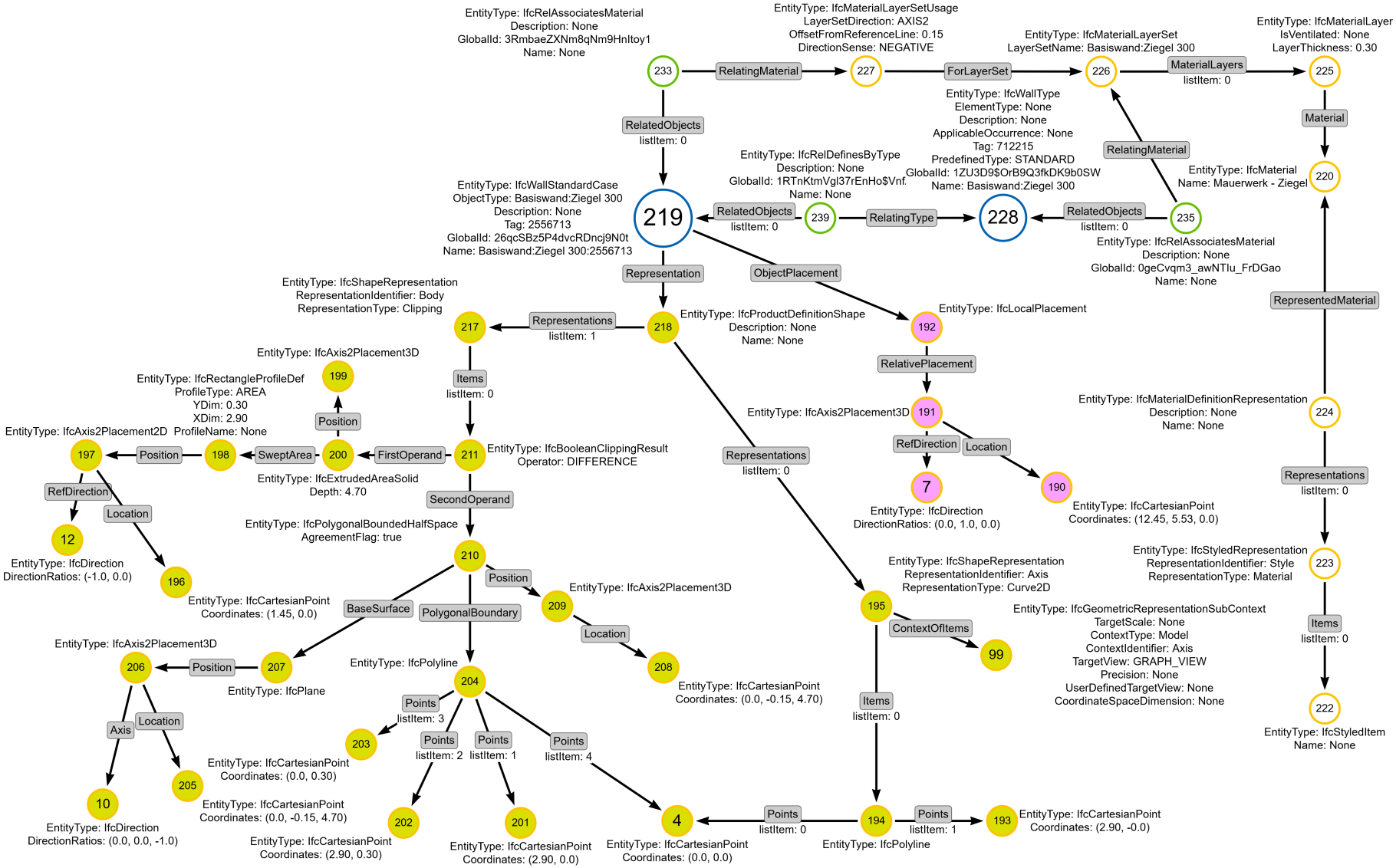


Abbildung 6.12: PushOut-Muster des Inkrements $\delta_{23,ARC}$

Abbildung 6.13: Klebe-Muster des Inkrements $\delta_{23,ARC}$

Die Anwendung des Inkrements in Cypher-Sprache ist in Anhang A.1.3 dokumentiert. Da die eingefügte Wand keine lastabtragende Funktion besitzt, erfolgt keine korrespondierende Änderung des Tragwerkssmodells.

6.3 Zusammenfassung und Einordnung

In diesem Kapitel wurde ein beispielhafter Prozess zwischen einem Architekten und einem Tragwerksplaner beschrieben. Nachdem der Architekt ein initiales Modell erstellt hat, konnte der Tragwerksplaner auf dessen Grundlage ein initiales Tragwerksmodell ableiten, welches anschließend für verschiedene statische Berechnungen herangezogen wurde. Als Ergebnis dieser Simulationen wurden anschließend die Stützenquerschnitte der Außenstützen modifiziert und die Beschaffenheit des korrespondierenden Inkrements in Abschnitt 6.2.1 erläutert. In ähnlicher Weise wurden die betroffenen Stützen anschließend im Architekturmodell modifiziert, wodurch das in Abschnitt 6.2.2 beschriebene Inkrement formuliert wurde. Darüber hinaus wurde in Abschnitt 6.2.3 exemplarisch beschrieben, welche Muster notwendig waren, um das Hinzufügen der neuen Innenwand im Architekturmodell zu beschreiben. Die durchgeführte Fallstudie verdeutlicht noch einmal die Mächtigkeit des entwickelten Ansatzes, zeigt aber auch die Sensitivität des Systems. Insbesondere bei den in Abschnitt 6.2.2 erläuterten semantischen Modifikationen wurde deutlich, dass auch marginale Änderungen an Gleitkommazahlen exakt übertragen werden. Eine solche Modifikation hat aber voraussichtlich nur in seltenen Fällen eine planerische Konsequenz und könnte durch geeignete Annahmen zum Runden solcher

Werte gelöst werden. Da es sich hierbei erneut um eine Interpretation der überwachten Datenmenge handelt, müsste dazu eine entsprechende Annahme getroffen werden, um eine solche Prozessierung spezifisch durchzuführen und die erzielten Ergebnisse damit weitergehend zu filtern.

Betrachtet man die beiden Modifikationen an den Stützenquerschnitten in Tragwerks- und Architekturmodell, sind gewisse Ähnlichkeiten in den Mustern zur Identifikation der veränderten Knoten erkennbar. Sollten derartige Inkremente häufiger in einem Austauschprozess synchronisiert werden, könnte diese Korrespondenz für weitergehende Überlegungen zur automatisisierten Anpassung disziplinfremder Graphrepräsentationen genutzt werden. Aus den dargestellten semantischen Modifikationen könnte beispielsweise bidirektional abgeleitet werden, dass es sich um einen rechteckigen Querschnitt handelt. Denkbar wäre daher, die im Inkrement $\delta_{12,TW}$ enthaltenen Informationen auf der Seite des Architekten weitergehend aufzubereiten, um direkten Rückschluss auf die notwendige Änderung des Architekturmodells zu erlangen und diese nach Freigabe durch den Nutzer automatisiert zu vollziehen. Gleichzeitig wird aber an dieser vergleichsweise einfachen Änderung deutlich, dass eine direkte Übertragung eines Inkrements auf ein anderes Modell selten bis nie funktionieren können wird.

Die behandelten Änderungen verdeutlichen einmal mehr, dass mit dem entwickelten System eine umfassende Grundlage für die Versionskontrolle und der Synchronisation von Modelländerungen konzipiert wurde. Erweitert man dieses System an geeigneten Stellen um zusätzliches Wissen über die Beschaffenheit der Daten oder über den repräsentierten Sachverhalt, ist eine weitergehende Automatisierung bestimmter Prozesse denkbar.

Kapitel 7

Diskussion

Dieses Kapitel fasst in Abschnitt 7.1 die erzielten konzeptionellen Ergebnisse zusammen. Abschnitt 7.2 stellt dar, inwiefern die aufgeworfenen Forschungsfragen beantwortet werden konnten, welche Limitationen die entwickelte Methodik aufweist und wie diese in zukünftigen Forschungsaktivitäten gelöst werden können. Das Kapitel schließt mit einer kurzen Zusammenfassung in Abschnitt 7.4.

7.1 Erzielte Funktionalitäten

Das vorgestellte System zur Versionsverwaltung von BIM-Modellen ermöglicht eine präzise Überwachung von Änderungen an Objekten innerhalb eines Modells, das die bisherigen Prinzipien der modellbasierten Kollaboration um eine inkrementelle Versionskontrolle erweitert. Zentral ist dabei die Betrachtung der in den Modellen enthaltenen Objektnetzwerke als Graphstrukturen. Um die entwickelte Methodik für möglichst viele Datenmodelle nutzen zu können, die den grundlegenden Prinzipien objektorientierter Datenmodellierung folgen, wurde ein flexibles und für zahlreiche Datenmodelle passendes Graph-Metamodell entwickelt. Durch die Verwendung der drei Knotentypen *Primärknoten*, *Sekundärknoten* und *Beziehungsknoten* konnte eine allgemeine Beschreibung gefunden werden, auf Grundlage derer die vorgestellten Mechanismen zur Abstraktion und Integration eines Aktualisierungsinkrements eine verlustfreie Übertragung von Modelländerungen ermöglichen. Die Anwendung des Systems wurde primär unter Berücksichtigung spezifischer Aspekte des IFC-Datenmodells und den zugehörigen Mechanismen von EXPRESS motiviert. Allerdings lässt sich die Methodik auf beinahe jede objektorientierte Datenstruktur anwenden, solange eine Teilmenge von Objekten mit eindeutigen, über verschiedene Versionen hinweg stabilen Identifikationsmerkmalen ansprechbar ist.

Während sich Kapitel 4 vorrangig auf die Erläuterung aller notwendigen Teilschritte zur Abbildung von BIM-Modellen als LPG-Graph, der Suche nach einem gemeinsamen Subgraph G_{MCS} sowie der Ableitung des Versionsinkrements konzentrierte, wurde die Methodik in Kapitel 5 im Kontext eines vollwertigen Versionskontrollsystems für BIM-Modelle weiterentwickelt. Hervorzuheben sind dabei besonders die Überlegungen zum Umgang mit divergierenden Modellversionen in verschiedenen Entwicklungszweigen sowie deren späterer Zusammenführung in konsolidierte Versionen eines Modells. Darüber hinaus wurden Prinzipien zur Kopplung von Modellkomponenten in einem interdisziplinären Kontext vorgestellt, die unter anderem zur direkten Evaluierung anvisierter Änderungen und der Benachrichtigung anderer Disziplinen dient, sofern ein Inkrement neben der Modifikation des zugehörigen Disziplinmodells auch disziplinübergreifende Auswirkungen

entfaltet. Hervorzuheben sind dabei weiterhin die hohe Flexibilität in der Ausgestaltung von Synchronisationszyklen sowie die allgemeine Einsetzbarkeit für unterschiedliche Datenrepräsentationen, die das Versionskontrollsystem verwalten kann. Die Anwendung der vorgestellten Konzepte konnte abschließend in Kapitel 6 erfolgreich für eine Interaktion zwischen Architektur- und Tragwerksplanung demonstriert werden.

7.2 Beantwortung der Forschungsfragen

7.2.1 Forschungsfrage 1

Wie können in der Softwareentwicklung etablierte Prinzipien der optimistischen Versionskontrolle auf planerische Prozesse im Bauwesen übertragen werden?

Betrachtet man die heute gängige Praxis in der Baubranche, sind dort bereits bestimmte Aspekte erkennbar, die den in Abschnitt 2.6 vorgestellten Überlegungen einer optimistischen Nebenläufigkeitsverwaltung in verteilten Systemen folgen. Verschiedene Disziplinen arbeiten derzeit überwiegend unabhängig von anderen Akteuren und tauschen ihre Planungsstände zu definierten Zeitpunkten in Form von monolithischen Modellen miteinander aus. Durch das strukturierte Erfassen von disziplinspezifischen Informationen in einem passenden Fachmodell wird der interdisziplinäre Informationsaustausch ermöglicht und gefördert. Für die Abstimmung zwischen den einzelnen Projektbeteiligten werden die einzelnen Modelle in ein Koordinationsmodell integriert. Mithilfe dieser gesamtheitlichen Darstellung werden Untersuchungen zur interdisziplinären Konsistenz vorgenommen. Festzuhalten ist daher, dass die wesentlichen erläuterten Grundprinzipien zur optimistischen Nebenläufigkeit bereits im BIM Level 2 Reifegrad in gewisser Ausprägung Anwendung finden.

Als signifikantes Defizit wurde aus den bisherigen Methoden allerdings herausgearbeitet, dass die von Änderungen betroffenen Objekte nicht direkt zugänglich gemacht werden können, sondern immer innerhalb eines monolithischen Modells erneut ausgetauscht werden. Betrachtet man etablierte Verfahren in der Softwareentwicklung, arbeiten auch dort einzelne Entwickler unabhängig voneinander und integrieren zu festgelegten Zeitpunkten ihre eigene Arbeit in gemeinsame Kollaborationsumgebungen. Als wesentlicher Unterschied existiert dort aber bereits die Möglichkeit, Modifikationen an Dateien inkrementell zu erfassen und diese anschließend transparent mithilfe eines Versionskontrollsystems zu überwachen. Dieser Baustein fehlt der modellbasierten Kollaboration im Bauwesen bisher und wurde durch die Verwendung graphbasierter Repräsentationen von BIM-Modellen im Rahmen dieser Arbeit vorgestellt. Basierend auf der generischen Beschreibung aller Objekte eines Modells als Knoten und ihren Beziehungen als Kanten wurde ein Ansatz entwickelt, der vorgenommene Änderungen allein auf der Analyse des Objektgeflechts ermittelt, ohne dabei auf die konkreten Bedeutungen einzelner Objekte einzugehen. Ebenso wurde die Ermittlung sowie die Anwendung von Inkrementen so allgemein wie möglich beschrieben, um die notwendigen Randbedingungen möglichst gering zu halten.

Gleichzeitig wurden Möglichkeiten für die Nutzer implementiert, um das Kontrollsystem präzise steuern zu können. Als Orientierung wurden hierfür die Befehle des Git-Systems analysiert und für die Belange der Versionskontrolle von BIM-Modellen erweitert. Damit erhält der Nutzer die volle Kontrolle über sämtliche Vorgänge, die mit dem Versionskontrollsystem abgebildet werden und kann insbesondere die Zeitpunkte zur Erstellung eines Versionsschritts oder der Synchronisierung mit dem zentralen Kollaborations-Hub individuell wählen. Neben der präzisen zeitlichen Steuerung wurden Befehle definiert, die den Umgang mit divergierenden Modellversionen in unterschiedlichen Zweigen sowie innerhalb des Gesamtsystems mit mehreren Disziplinen und Nutzern ermöglichen. Die vorgestellten Untersuchungen und Entwicklungen haben das Potenzial für die weitere Verbesserung kollaborativer Methoden im Bauwesen bestätigt und deren Anwendung exemplarisch für herstellerneutrale Datenmodelle demonstriert. Weiteres Potenzial ist dabei aber insbesondere für weitere automatisierbare Prozesse denkbar, die Versionsinkremente als Eingangsgröße nutzen könnten.

Vergleicht man diese Überlegungen mit fortgeschrittenen Funktionalitäten etablierter Plattformen zur Quellcodeverwaltung, fällt auf, dass weiteres Potenzial in der automatisierten Auswertung der Versionsinkremente zu erwarten ist. Insbesondere große Anbieter von Plattformen zur Quellcodeverwaltung wie GitHub bieten heute umfangreiche Funktionen an, um nachgelagerte Prozesse im Falle neuer Commits anzustoßen. Dazu zählt beispielsweise die automatische Bereitstellung des Quellcodes als kompilierte Repräsentation oder die automatische Ausführung von Unit-Tests. Solche Überlegungen stecken im Bauwesen derzeit noch in einem frühen Stadium. Als Grund sind hier einerseits die vielfältigen Möglichkeiten zur Abbildung planerischer Informationen in den verfügbaren Datenmodellen und der damit verbundenen Komplexität der Informationsauswertung und andererseits die vergleichsweise geringe Marktnachfrage nach solchen Lösungen zu nennen. Softwareentwicklung und insbesondere deren flexible Auslieferung auf unterschiedliche Plattformen spielt mittlerweile eine enorme Rolle in diesem Geschäftsbereich. Daher kann hierbei auf ein breites Portfolio verschiedener Anbieter zurückgegriffen werden, welche die Orchestrierung solcher Automatisierungen begleiten oder gar vollständig übernehmen. So bleibt zu hoffen, dass die in dieser Arbeit dargelegten Ansätze zur objektbasierten Versionskontrolle von BIM-Modellen passende Grundlagen für zukünftige Entwicklungen weitergehender Automatisierungen bieten kann. Wie beschrieben sollte der Input menschlicher Intelligenz dabei nach wie vor berücksichtigt werden. Routinen zur direkten Auswertung von Inkrementen können aber helfen, wiederkehrende und gut abgrenzbare Änderungen zukünftig zumindest teil-automatisiert über Disziplingrenzen hinweg anwenden zu können.

7.2.2 Forschungsfrage 2

Inwiefern eignen sich Konzepte der Graphtheorie zur Beschreibung der in BIM-Modellen vorliegenden hochvernetzten Informationen und einer verlustfreien Übertragung von Modell-Veränderungen?

Die durchgeführten Untersuchungen haben belegt, dass die Konzepte von LPG-Graphrepräsentationen geeignet sind, die in einem BIM-Modell enthaltenen Informationen verlustfrei zu speichern und für Mechanismen der Versionsverwaltung zu nutzen. Hierfür wurde ein eigenes Graph-Metamodell entwickelt, das die gewonnenen Erkenntnisse aus den verschiedenen untersuchten Datenmodellen bündelt und die Anwendbarkeit der vorgestellten Methoden für eine Vielzahl von Repräsentationen sicherstellt. Unterschieden wird hierbei lediglich in drei verschiedene Knotenarten. *Primärknoten* werden genutzt, um Instanzen im Graph zu modellieren, die über ein eindeutiges, persistentes Merkmal verfügen. *Sekundärknoten* dienen zur Abbildung von Informationen, die kein eindeutiges Merkmal aufweisen. Deren Lage im Graph kann allerdings immer durch Pfade bestehend aus einem Primärknoten und mehreren Sekundärknoten beschrieben werden. Geeignete Pfade können dabei auf Basis der topologischen Sortierung gemäß den Grundlagen zu DAG-Graphen ermittelt werden. Als dritte Knotenart bilden *Beziehungsknoten* das geeignete Medium zur Umsetzung objektifizierter Beziehungen. Diese sind insbesondere für Datenmodelle relevant, die *viele-zu-viele* Beziehungen zulassen.

Die Einsatz des definierten Graph-Metamodells wurde vorrangig mit Instanzdaten demonstriert, die dem IFC-Datenmodell oder verallgemeinert den Grundprinzipien der Normenreihe ISO 10303 mit der SPF-Serialisierung folgen. Datenmodelle, deren Instanzen mit den Serialisierungssprachen XML oder JSON beschrieben werden, können ebenfalls mit dem gewählten Graph-Metamodell dargestellt werden. Die gewählte Repräsentation der in einem BIM-Modell enthaltenen Informationen als LPG-Graph ist daher als flexibel einsetzbar zu bewerten. Durch die Übersetzung wurden alle Phänomene eliminiert, die durch die Serialisierung von Objektstrukturen in eine textuelle Form eingetragen wurden und die die Verwendung textbasierter Versionskontrollsysteme für die Verwaltung von BIM-Modells verhindern. Zu diesen zählen einerseits die verwendete Export-Reihenfolge der in einem Modell enthaltenen Objekte sowie andererseits die willkürliche Nummerierung von Instanzen, um in der serialisierten Form Beziehungen abbilden zu können. Zur Repräsentation der Informationen eines Modells wurden die Prinzipien von LPG-Graphen genutzt. Diese Graphen folgen den grundlegenden Prinzipien der Graphentheorie und ermöglichen zusätzlich die Vergabe von Labels sowie Attributen an Knoten und Kanten. Die in Auszügen vorgestellte Anfragesprache Cypher konnte sowohl für die Erstellung als auch für die anschließende Interaktion mit den Graphen erfolgreich verwendet werden.

Die Anwendung wesentlicher graphtheoretischer Grundlagen auf das Problem der Versionskontrolle von BIM-Modellen ist insgesamt positiv zu bewerten und hat maßgeblich zur Erarbeitung passender Lösungsstrategien für die aufgeworfenen Fragestellungen beigetragen. Herausfordernd gestaltete sich allerdings an einzelnen Stellen der Transfer der in der Literatur allgemein beschriebenen Konzepte zu Aspekten der Graphtransformation und deren spezifischer Anwendung auf LPG-Graphen. Insbesondere die Spezifika, die sich aus der Unterscheidung zwischen topologischen, semantischen oder kombinierten Mustern sowie aus der Betrachtung von Knoten ohne eindeutige Merkmale ergaben, erschwerten zum Teil den direkten Transfer allgemeiner Konzepte auf das spezifische Problem der Versionsverwaltung semantischer Graphen.

Wie beschrieben stellte die möglichst generalisierte Repräsentation von Instanzen unterschiedlicher Datenmodelle ein zentrales Ziel der Untersuchungen dar. Dieses Ziel wurde durch das gewählte Graph-Metamodell erreicht, geht aber zu Lasten der syntaktischen Überwachung, inwiefern die im Graph modellierten Informationen konform zu dem zugrundeliegenden Datenmodell sind. Das Graph-Metamodell wurde bewusst sehr generell gewählt und reflektiert dabei die minimal notwendigen Strukturen, die notwendig sind, um die Lage jedes Knotens entweder durch ein persistentes Merkmal oder durch einen passenden Pfad zu beschreiben. Der gewählte Ansatz verlässt sich vollständig darauf, dass die exportierenden BIM-Autorenwerkzeuge ein Objektnetzwerk erzeugen, das zu dem zugrundeliegenden Datenmodell inhaltlich und syntaktisch passt. In der aktuellen Ausprägung ist das Versionskontrollsystem nicht imstande, mögliche fehlerhafte Instanzen zu detektieren, die nicht konform zum jeweiligen Datenmodell sind. Zusätzliche oder fehlerhaft belegte Knoten- und Kantenattribute oder unzulässige Beziehungen zwischen Knoten können durch die fehlende Repräsentation der Datenmodelle im Graph-Metamodell nicht erkannt werden.

Ist eine Konformitätsprüfung direkt im Graphspeicher gewünscht, wäre eine Erweiterung des Graph-Metamodells um weitere Knoten- und Kantentypen sowie der Abbildung verschiedener Bedingungen notwendig. Hierfür steht eine breite Palette an technologischen Lösungen im Kontext semantischer Graphen bereit. Im verwendeten Graphdatenbanksystem neo4j können beispielsweise so genannte *Constraints* definiert werden, die das Belegen bestimmter Attribute an Knoten und Kanten auf Basis der verwendeten Labels erzwingen. Darüber hinaus kann die Verwendung bestimmter Kantentypen weitergehend beschränkt werden, indem die zulässigen inzidenten Knotentypen vorab definiert werden. Eine vollständige Abbildung der Datenmodelle würde folglich eine bessere Überwachung ermöglichen, ob die im Graph enthaltenen Informationen tatsächlich allen Vorgaben des Datenmodells entsprechen. Diese gehen aber klar zu Lasten der generischen Betrachtung. Ist zu Beginn allerdings absehbar, dass nur Instanzen bestimmter Datenmodelle im Versionskontrollsystem behandelt werden sollen, können die genutzten Graphspeicher ohne Auswirkung auf die entwickelte Methodik erweitert werden.

In den durchgeführten Experimenten ist zudem aufgefallen, dass insbesondere die Prozessierung von Modellen mit detaillierten und komplexen geometrischen Repräsentationen gravierende Herausforderungen in den Ausführungszeiten mit sich bringen. Hierfür ist die gewählte Repräsentation aller Modellinhalte als Graph nur bedingt sinnvoll. Insbesondere für Geometrien, die basierend auf ihren Oberflächen durch Knoten und Facetten beschrieben werden, korreliert die Anzahl an Knoten und Kanten im resultierenden Graph mit der geometrischen Detaillierung. Je genauer Geometrien approximiert werden, desto größer wird die Zahl der Koordinaten, welche die Oberfläche beschreiben. Der vorgestellte Ansatz erfasst und übermittelt auch hier sämtliche Änderungen in Form von semantischen oder topologischen Änderungen. Diese Änderungen sind allerdings in vielen Fällen aus Sicht eines Nutzers nicht aufschlussreich. Ursächlich hierfür sind einerseits numerische Phänomene in der Abbildung von Gleitkommazahlen und andererseits die Tatsache, dass eine erneute Vermaschung einer Oberfläche abhängig von den gewählten Randbedingun-

gen zu veränderten Koordinatenwerten führen kann. Letztgenannter Vorgang muss aber nicht zwingend zu einer Änderung der Dateninterpretation führen, sodass der Austausch solcher Inkremente zwar strikt jegliche Modifikation übermittelt, aber diese möglicherweise keine neuen Informationen für die Nutzung des Modells auf Empfängerseite bieten.

Sofern man von der gewählten bijektiven Abbildung jeder Instanz auf einen Knoten und jeder Beziehung auf eine Kante im Graph abweichen möchte, können verschiedene Überlegungen angestellt werden, die den Abgleich zweier Modellversionen effizienter gestalten könnten. Einige sind in Abb. 7.1 illustriert und werden im Folgenden kurz skizziert.

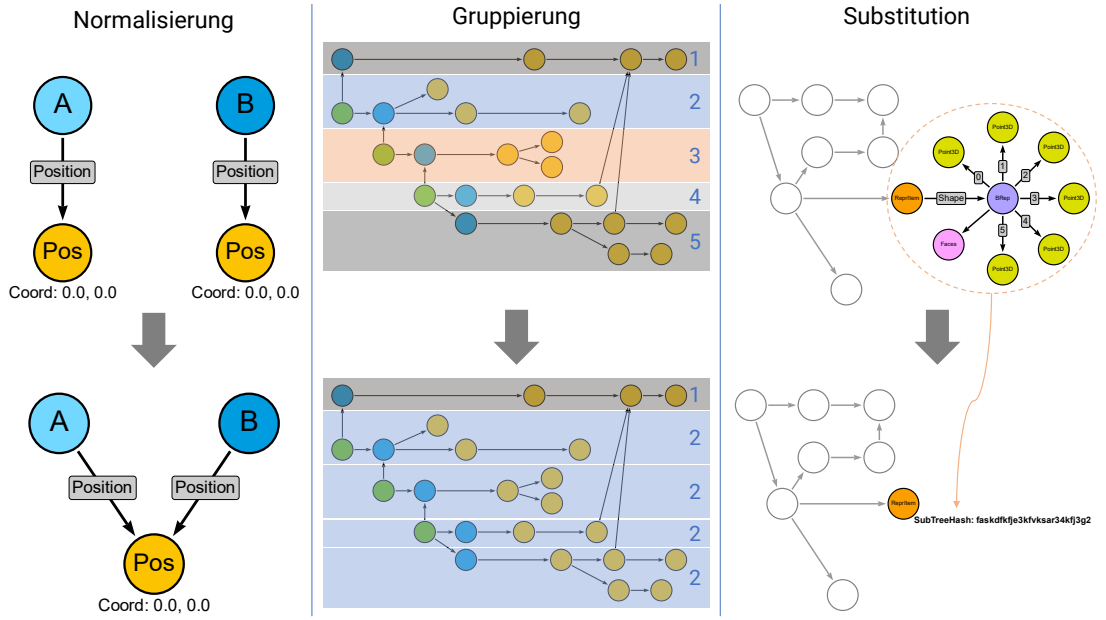


Abbildung 7.1: Optimierungen zur effizienteren Ermittlung des gemeinsamen Subgraphs G_{MCS}

Normalisierung der zu traversierenden Graphen

Als erste Optimierung ist denkbar, die Graphen grundsätzlich vor der Traversierung zu normalisieren und redundante Informationen zu entfernen. Eine solche vorgelagerte Prozessierung setzen beispielsweise Shi et al. (2018) ein. Vorteilhaft ist dabei, dass die Graphen dadurch eine kompaktere Form aufweisen. Nachteilig ist gleichzeitig festzuhalten, dass die originäre Objektstruktur nicht mehr bijektiv im Graph abgebildet wird und damit die in Abschnitt 4.1 zugehörige Randbedingung verletzt wird. Alternativ könnte die Forderung nach möglichst kompakten, normalisierten Repräsentationen aber auch als Anliegen an die modellerzeugenden Schnittstellen bestehender Softwareprodukte formuliert werden.

Gruppierung und parallele Prozessierung mehrerer Teilgraphen

Als zweite Maßnahme ist denkbar, die Graphstrukturen vor und während der Traversierung weitergehend zu analysieren und dadurch mehrere Traversierungen parallel durchzuführen. Insbesondere bei BIM-Modellen, die das IFC-Datenmodell instanziieren, ist in zahlreichen Fällen aufgefallen, dass einzelne Modellkomponenten lediglich Verweise

auf einen gemeinsam genutzten geometrischen Kontext sowie relative Positionierungen aufweisen. Alle anderen Knoten und Kanten, die innerhalb des rekursiven Durchlaufens einer Modellkomponente traversiert wurden, dienen alleinig als Informationsträger für die aktuell untersuchte Komponente. Denkbar ist daher, vor der Traversierung eine Gruppierung im Graph vorzunehmen, um disjunkte Teilgraphen zu identifizieren und diese dann parallel zu traversieren. Dies zeigt schematisch die mittlere Spalte in Abb. 7.1. Nach der Traversierung des oben angeordneten blauen Primärknotens und aller anschließenden in gelb dargestellten Sekundärknoten können die anderen Modellkomponenten parallel verarbeitet werden, da sie lediglich Verweise auf bereits traversierte Knoten haben oder zu ihnen gänzlich eigenständige Subgraphen adjazent sind.

Substitution von Teilgraphen

Neben der Normalisierung und Gruppierung ist als dritte Maßnahme die Substitution bestimmter Teilgraphen möglich. Hierzu sind induzierte Teilgraphen $G' \subseteq G$ von Interesse, die durch einen einzigen Knoten substituierbar sind. Die Substitution selbst lässt sich durch eine Graphtransformation beschreiben, bei der im Mustergraph L das zu substituierende Muster spezifiziert wird und der Ersetzungsgraph R lediglich einen Knoten mit einem Substitutionsattribut enthält (siehe hierzu ebenfalls Han et al., 2023). Die Substitution würde dann vor der Traversierung erfolgen, um wie im Falle der Normalisierung die Anzahl von Knoten und Kanten im Graph ohne Informationsverlust zu verringern. Um substituierbare Teilgraphen zu erkennen, können entweder generische Suchen auf den Graphen ausgeführt oder zusätzliches Vorwissen über die im Graph enthaltenen Informationen genutzt werden. Die durchgeführten Experimente mit dem IFC-Datenmodell haben beispielhaft gezeigt, dass in vielen Fällen die überwiegende Anzahl von Instanzen für die Beschreibung geometrischer Informationen benötigt werden. Auch wenn der entwickelte Ansatz vorgenommene Modifikationen an geometrischen Repräsentationen zuverlässig erkennt und kodiert, ergeben sich zwei Herausforderungen. Einerseits können aus den ermittelten Inkrementen häufig keine aussagekräftigen Folgerungen über die tatsächliche Auswirkung auf die geometrische Form eines Bauteils getroffen werden. Andererseits skaliert die Laufzeit zur Ermittlung des gemeinsamen Subgraphs mit der Anzahl zu traversierender Elemente und kann dabei im Falle großer Teilgraphen für die Abbildung der Geometrien zu erheblichen Performance-Problemen führen.

Daher könnte als mögliche Verbesserung vorgesehen werden, geometrische Repräsentationen zukünftig in einem separierten, dafür optimierten Geometriespeicher zu verwalten. Solche Überlegungen sind teilweise in Veröffentlichungen zur Verwendung von Linked-Data-Repräsentationen für Bauwerksinformationen bereits dokumentiert worden (Bonduel, 2021). Die Bauteilgeometrien werden dabei außerhalb des Graphs als *obj*-Dateien vorgehalten. Diese beinhalten explizite Geometrierepräsentationen der Bauteiloberflächen im Sinne einer *Boundary-Representation* (BRep)-Geometrie. Zu überlegen ist daher eine weitere Aufteilung des Versionskontrollsystems in zwei dedizierte Speicher für semantische und geometrische Informationen. Abb. 7.2 illustriert eine mögliche Erweiterung der Speicherstruktur hinsichtlich einer verbesserten Verwaltung geometrischer Repräsentationen.

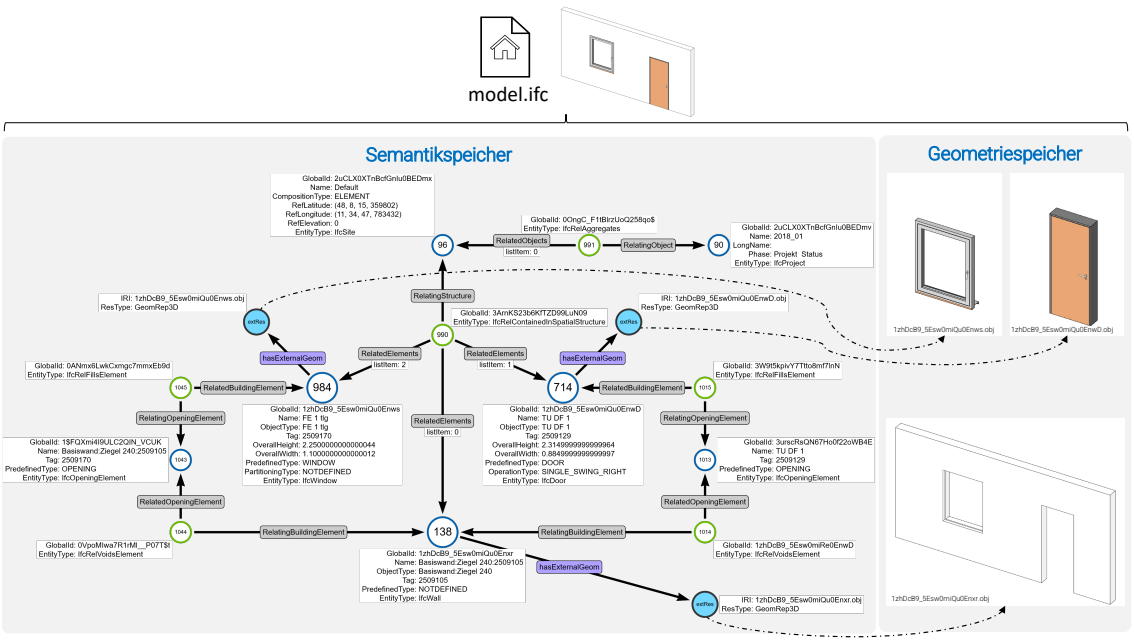


Abbildung 7.2: Separieren semantischer und geometrischer Informationen in ein zweiteiliges Versionskontrollsystem

Um die Speicherung von Ressourcen außerhalb des Graphs zu ermöglichen, müsste das Graph-Metamodell um geeignete Elemente erweitert werden. Die Entkopplung der Geometrien wird im illustrierten Beispiel durch einen neuen Knotentyp abgebildet, der für die Speicherung von Verweisen auf externe Ressourcen dient. In diesen Knoten können dann entsprechende Verweise auf Dateien vorgenommen werden, die die geometrischen Informationen zu einem Bauteil beschreiben. Erreicht die Traversierung diese Knoten, können entsprechende Substitutionswerte an den Knoten verwendet werden, um über mögliche Änderungen in den geometrischen Ausgestaltungen zu informieren. Zudem müsste dem entwickelten Datenmodell zur Beschreibung der Graphmodifikationen passende Klassen zur Erfassung geometrischer Modifikationen hinzugefügt werden (siehe hierzu auch Abb. 4.12). Diese hängen allerdings maßgeblich von den gewählten Geometriespeichern und den gewählten Verfahren zum Abgleich zweier Geometrien ab, sodass diese als Gegenstand zukünftiger Forschungsaktivitäten eingeordnet wird. Dabei sollten insbesondere geometrische Repräsentationen betrachtet werden, die implizite sowie prozedurale Verfahren nutzen und geometrische Formen beispielsweise in Abhängigkeit von Achsen oder anderen Bauteilen beschreiben. Solche Formen sind insbesondere bei der Verwaltung von BIM-Modellen zu erwarten, die eine Modellierung von Infrastrukturanlagen beinhalten.

Da die skizzierten Überlegungen das Grundprinzip der bijektiven Abbildung der objektorientierten Strukturen auf eine entsprechende Graphrepräsentation verlassen, wurden diese Ansätze bisher nicht in der Tiefe verfolgt, da die allgemeine Anwendbarkeit der Methodik dadurch stark geschwächt werden würde. Inwiefern die beschriebenen Optimierungen aber für spezifische Situationen oder Datenmodelle sinnvoll sein können, sollte in zukünftiger Forschung weitergehend untersucht werden. Würden Mischformen aus der bisherigen generischen Beschreibung aller Instanzen im Graph und substituierten oder

hybriden Speichern genutzt werden, müsste darüber hinaus sichergestellt werden, dass jeder Akteur eines Projektes mit der gleichen Konfiguration arbeitet.

7.2.3 Forschungsfrage 3

Wie können inkrementelle Änderungen zwischen Modellversionen abstrahiert und unter Berücksichtigung bestehender Standards interdisziplinär ausgetauscht werden?

Wie Kapitel 4 dargelegt hat, werden die Inkremente auf Basis zweier Modellversionen ermittelt, die aus dem verwendeten BIM-Autorenwerkzeug in ein herstellerneutrales Format exportiert wurden. Gegenüber einer ständigen Überwachung der Entwurfssoftware und den dort durchgeführten Transaktionen ist dieser Ansatz als allgemeingültiger einzustufen und kann ohne spezifische Implementierungen für ein eingesetztes Autorensystem direkt genutzt werden, sofern bestehende Export-Schnittstellen zur Verfügung stehen. Vorgenommene Modifikationen werden ausgehend von zwei gegebenen Versionen eines BIM-Modells ermittelt. Nach der Übersetzung beider Modellversionen in ihre Graphrepräsentationen wird mithilfe eines rekursiven Verfahrens untersucht, welche Knoten aus dem initialen und aktualisierten Graph als äquivalent eingestuft werden können. Für die Äquivalenz wird bei Primärknoten das persistente, eindeutige Merkmal verwendet, wohingegen Sekundärknoten basierend auf ihrer Lage im Graph bewertet werden. Hierfür werden die Attribute der Kante genutzt, anhand derer im aktuellen Rekursionsschritt traversiert wird, und in Kombination mit dem zu dieser Kante inzidenten Entitätstyp des Knotens bewertet. Sind zwei Knoten als äquivalent eingestuft, wird eine Äquivalenzkante hinzugefügt und anschließend ein Vergleich der Knotenattribute durchgeführt. Sind Abweichungen festzustellen, werden diese als semantische Modifikation erfasst.

Nach Abschluss des rekursiven Verfahrens wird von den nun verbundenen Graphen der gemeinsame Subgraph G_{MCS} abgeleitet und die topologischen Modifikationen abgeleitet. Knoten und Kanten, die nicht Teil des gemeinsamen Teilgraphs sind, müssen folglich als topologische Modifikation behandelt werden. Deren Erfassung teilt sich in drei Muster auf. Das PushOut-Muster beschreibt den Teilgraphen, der entweder gelöscht oder neu hinzugefügt werden muss. Das Kontext-Muster spezifiziert Knoten des gemeinsamen Subgraphs G_{MCS}, die für eine erfolgreiche Verknüpfung der zu löschenden oder einzufügenden Teile an die unveränderten Informationen notwendig sind. Schließlich beschreibt das Klebe-Muster, wie Knoten des PushOut-Musters mit Knoten des Kontext-Musters mit Kanten verknüpft werden. Die drei Muster können anschließend in die DPO-Notation überführt werden und sind damit vollständig kompatibel mit verschiedenen Graphersetzungssystemen, die Double-Push-Out-Operationen auf semantischen Graphen ausführen können. Ein vollständiges Versionsinkrement ergibt sich schließlich aus allen ermittelten semantischen Modifikationen sowie den notwendigen topologischen Transformationen, um die initiale Version in die aktualisierte Version zu überführen. In der beschriebenen Methodik wird die notwendige topologische Transformation in einer einzigen Graphtransformationsregel erfasst. Denkbar wäre aber für zukünftige Erweiterungen auch, diese Regel

in mehrere Teil-Transformationen aufzuteilen, um beispielsweise die Selektion einzelner Änderungen des Inkrements im Falle eines Versions-Mergings ermöglichen zu können.

Neben der Methodik zur Ermittlung der Inkremente wurde ein Ansatz erarbeitet, der Nutzern präzise Möglichkeiten zur Erstellung und Synchronisation von Inkrementen mit anderen Projektbeteiligten an die Hand gibt. Darüber hinaus wurde dargelegt, inwiefern sich die ermittelten Inkremente für die parallele Arbeit in verschiedenen Entwicklungszweigen eignen und wie mit möglicherweise auftretenden Konflikten umgegangen werden kann. Der Austausch der Inkremente mit anderen Projektbeteiligten wird über einen Kollaborations-Hub realisiert, bei dem neben dem Informationsaustausch auch weitergehende Funktionalitäten zur interdisziplinären Verknüpfung von Objekten verschiedener Fachsichten vorgesehen sind. Um die Kompatibilität mit derzeitigen Schnittstellen sicherzustellen, kann der Nutzer die Graphen zudem zu jedem Zeitpunkt verlustfrei zurück in eine serialisierte Struktur übersetzen.

Die entwickelten Konzepte haben sich als funktional und flexibel gezeigt. Wie aber auch in der zuvor beantworteten Forschungsfrage ist es nicht vorgesehen, die Inhalte eines Inkrements auf die Konformität hinsichtlich möglicher Vorgaben des zugrundeliegenden Datenmodells zu überprüfen, das für die Kodierung der planerischen Informationen im BIM-Modell herangezogen wird. Erneut ist hierzu festzuhalten, dass für die Aufgabe der Erstellung schemakonformer Repräsentationen primär die erzeugenden Systeme verantwortlich sind. Ebenso ist es schwierig, mögliche Auswirkungen von Modifikationen auf andere Disziplinen in einer Weise vorherzusagen, die eine umfassende Automatisierung eingehender disziplinfremder Inkremente erlauben würde. Wie geschildert können bei der Nutzung eines Verknüpfungsgraphs interdisziplinäre Abhängigkeiten zwischen Objekten im Kollaborations-Hub modelliert werden, die insbesondere bei der Modifikation oder dem Entfernen bereits ausgetauschter Objekte Unterstützung bieten können. Werden hingegen neue Objekte zu einem Disziplinmodell hinzugefügt, müssten mögliche interdisziplinäre Abhängigkeiten aber erst ermittelt werden. Grundsätzlich besteht die Möglichkeit, solche Abhängigkeiten zusammen mit dem Inkrement auszutauschen, allerdings erfordern diese Überlegungen abermals das Hinzuziehen zusätzlichen Domänenwissens, was in gewissem Umfang die allgemeine Anwendbarkeit des entwickelten Systems schmälert.

7.2.4 Forschungsfrage 4

Worin bestehen Limitationen eines objektbasierten Versionskontrollsystems für BIM-Modelle?

Limitationen aufgrund der gewählten Traversierungsstrategie zur Ermittlung eines gemeinsamen Subgraphs

Wie in Abschnitt 4.5 erläutert erfolgt die Betrachtung zweier Knoten $v_{init} \in V_{init} - v_{updt} \in V_{updt}$ vorrangig auf Grundlage jener Attribute, die im aktuellen Traversierungsschritt an den betrachteten Kanten vorhanden sind. Wenngleich das Verfahren vorgenommene

Änderungen zuverlässig erkennt, können in Einzelfällen die erkannten gemeinsamen Subgraphen von ihrer maximal möglichen Größe abweichen. Zur Veranschaulichung der Limitationen sei das in Abb. 7.3 gegebene Beispiel diskutiert, in welchem zwei Bauteile (eine Instanz von *IfcWall* und *IfcWindow*) mit einer anderen Entität (hier *IfcSite* durch einen Relationsknoten verknüpft werden.

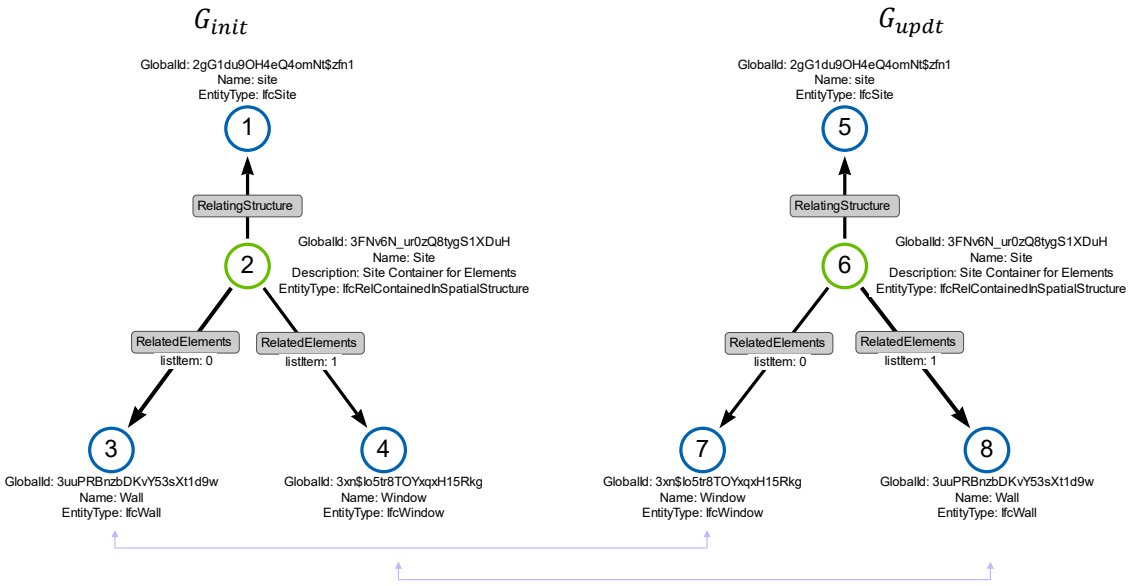
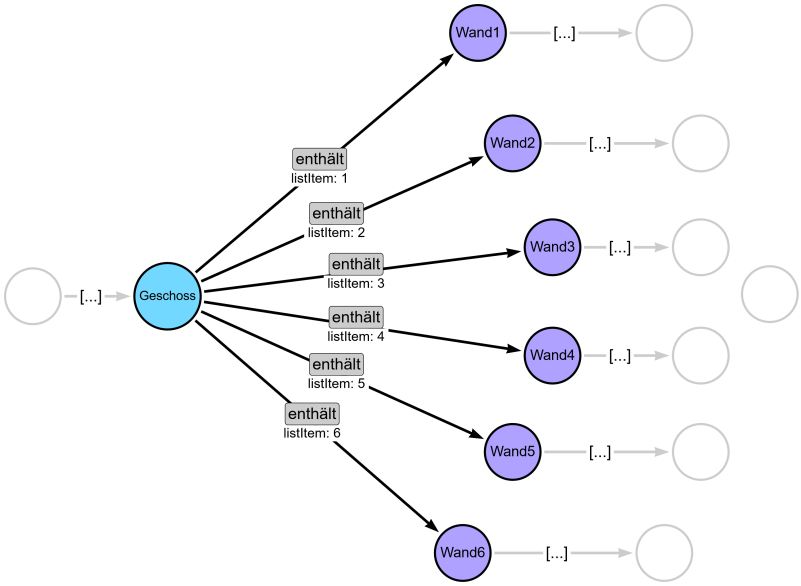


Abbildung 7.3: Limitation aufgrund der gewählten Äquivalenzkriterien

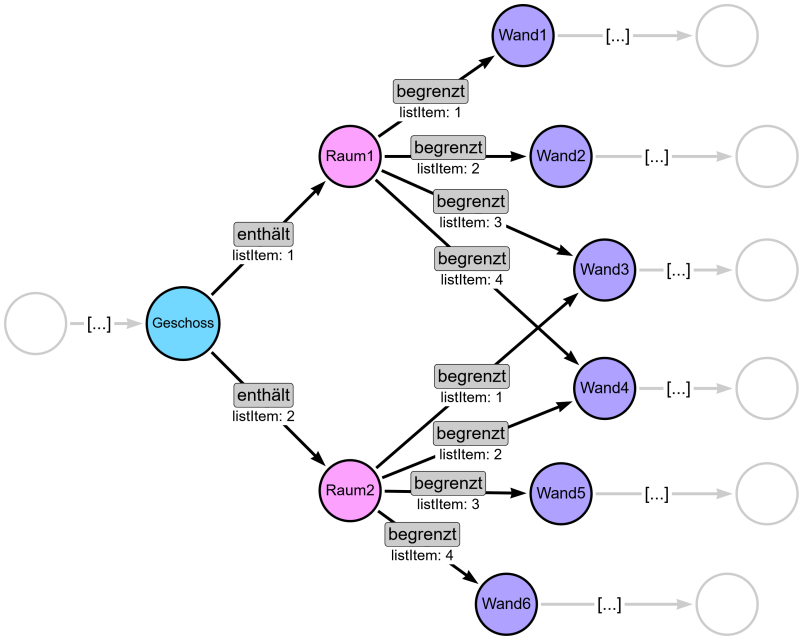
Gemäß den in Abschnitt 4.5 festgelegten Kriterien werden zwei Knoten dann als äquivalent betrachtet, wenn sie sowohl über den gleichen Relationstyp und die gleiche Listenposition verfügen sowie dieselbe Entität modellieren. Im dargestellten Beispiel würde die Traversierung daher versuchen, die Knotenpaare $3 - 7$ sowie $4 - 8$ in Korrespondenz zu bringen. Da diese allerdings unterschiedliche Werte im Attribut *EntityType* aufweisen, würde hier eine topologische Modifikation detektiert werden und im Anschluss einerseits das Löschen beider Entitäten aus dem Initialgraph G_{init} und das Einfügen zweier neuer Entitäten im aktualisierten Graph G_{updt} notwendig werden. Dieses Vorgehen würde schlussendlich die vorgenommenen Änderungen zwischen den beiden Modellzuständen richtig und verlustfrei beschreiben. Betrachtet man allerdings das Attribut *GlobalId* aller Knoten, so fällt auf, dass im Beispiel die Instanzen *IfcWall* und *IfcWindow* in umgekehrter Reihenfolge mit der Instanz von *IfcRelContainedInSpatialStructure* aggregiert wurden. Anstatt des vollständigen Entfernens und anschließendem Wiedereinfügens der Objekte wäre eine wesentlich aufwandsärmere Herangehensweise, lediglich die Kantenattribute entsprechend zu modifizieren. Im dargestellten Beispiel wäre dieses Phänomen durch eine Erweiterung der Entscheidungsmerkmale für die Knotenäquivalenz zu lösen.

Ähnliche Ergebnisse resultieren aus Situationen, bei denen ein Objekt in seiner räumlichen Zugehörigkeit lediglich verschoben wurde oder neue hierarchische Elemente zur Gruppierung von Inhalten hinzugefügt werden. Diese könnten nicht durch eine einfache Anpassung der Äquivalenzmerkmale verbessert werden, sondern würden umfangreichere

topologische Analysen benötigen. Abb. 7.4 illustriert eine beispielhafte Situation, bei der zwischen den untersuchten Versionen ein neuer räumlicher Container zur Modellierung zweier Räume hinzugefügt wurde. Neu hinzugekommen sind die zwei magentafarbenen Knoten, die zudem die Änderung diverser Kanten nach sich gezogen hat. Es wird außerdem angenommen, dass die Wände unverändert sind.



(a) Initialer Graph



(b) Aktualisierter Graph

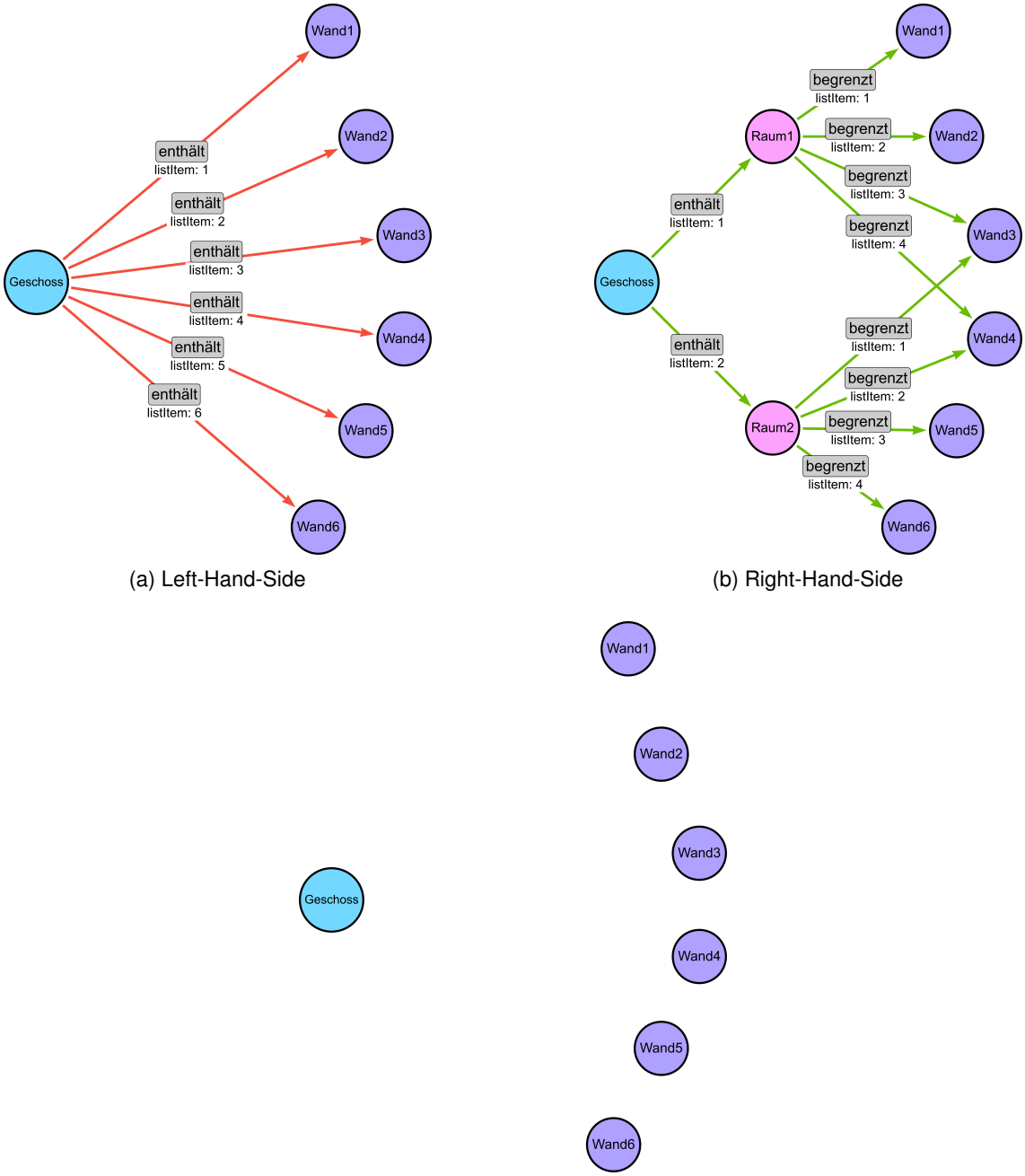
Abbildung 7.4: Beispielhafte Situation, bei der die Traversierungsstrategie unoptimale Äquivalenzen detektiert

Die entwickelten Algorithmen würden bei der Analyse der zum Geschoss adjazenten Knoten keine Äquivalenzen erkennen und die Traversierung hier stoppen. In der Folge

würde eine Transformationsregel abgeleitet werden, die einerseits alle Wände entfernt und anschließend die zwei Raumknoten sowie die Wände einfügt. Wie zuvor würde diese Transformation den Ausgangszustand ohne Informationsverluste in den gewünschten Zielzustand überführen. Die im Inkrement kodierten Änderungen sind allerdings deutlich umfangreicher als eigentlich notwendig, da der ermittelte gemeinsame Subgraph G_{MCS} durch die gewählte Traversierungsstrategie nicht die maximal mögliche Größe annimmt. Abb. 7.5 zeigt eine alternative Form, die lediglich das Einfügen der neuen Raumknoten sowie die Modifikation der Kanten beinhaltet.

Diese Beispiele zeigen Grenzen des gewählten Verfahrens zur Ermittlung der unveränderten Bestandteile in zwei Graphen. Auch bei dieser Situation kann die Berücksichtigung von zusätzlichem Vorwissen die Prozessierung unterstützen, was erneut die Generalisierbarkeit des Ansatzes und dessen Anwendung auf unterschiedliche Datenmodelle limitieren würde. Die geschilderte Problematik tritt allerdings nicht nur bei dem Einfügen neuer Strukturen zwischen unveränderten Teilgraphen auf, sondern entfaltet sich beispielsweise auch in Fällen, bei denen Bauteile einem neuen System zugeordnet werden. Wird ein Bauteil aus dem Erdgeschoss aus einem Stockwerk in eine andere räumliche Einheit verschoben, würde auch diese Änderung mithilfe eines vollständigen Entfernens und erneuten Einfügens ausgeführt werden.

Abhilfe könnte auch hier eine vorgelagerte Analyse des gesamten Graphs sein, bei dem beispielsweise nur die Primär- und Beziehungsknoten betrachtet werden. Unter Annahme persistenter Identifikationsmerkmale kann deren Übereinstimmung als Startpunkt für eine topologische Analyse sein, um mögliche Änderungen in der Anordnung aller Objekte frühzeitig zu erkennen und diese Information anschließend in die Traversierung einfließen zu lassen.



(a) Left-Hand-Side

(b) Right-Hand-Side

(c) Interface

Abbildung 7.5: Kompakte Form der notwendigen Transformation als DPO-Notation

Umgang mit Sekundärknoten, die die Charakteristik von Beziehungsknoten einnehmen

Bei dem Studium verschiedener Exportresultate gängiger BIM-Autorenwerkzeuge sind weitere Besonderheiten aufgefallen, die mit der aktuellen Traversierungsstrategie noch nicht abgedeckt sind. Beispielhaft seien dafür die IFC-Entitäten *IfcMaterialDefinitionRepresentation* und *IfcPresentationLayerAssignment* herausgegriffen, wovon erstgenannte

im Detail behandelt werden soll. Die Definition dieser Entität im IFC-Datenmodell ist in Algorithmus 7.1 dargestellt [1].

Algorithmus 7.1: Definition der Entität IfcPresentationLayerAssignment im IFC-Datenmodell

```
ENTITY IfcMaterialDefinitionRepresentation;
    Name:                   OPTIONAL IfcLabel;
    Description:            OPTIONAL IfcText;
    Representations:        LIST [1:?] OF IfcRepresentation;
    RepresentedMaterial:    IfcMaterial;
END_ENTITY;
```

Eine Instanz dieser Entität war Bestandteil des PushOut-Musters, das in Abschnitt 6.2.3 für das Einfügen der Wand in das Architekturmodell erläutert wurde. Der im Folgenden behandelte Knoten sowie die direkt adjazenten Nachbarn sind in Abb. 7.6 dargestellt.

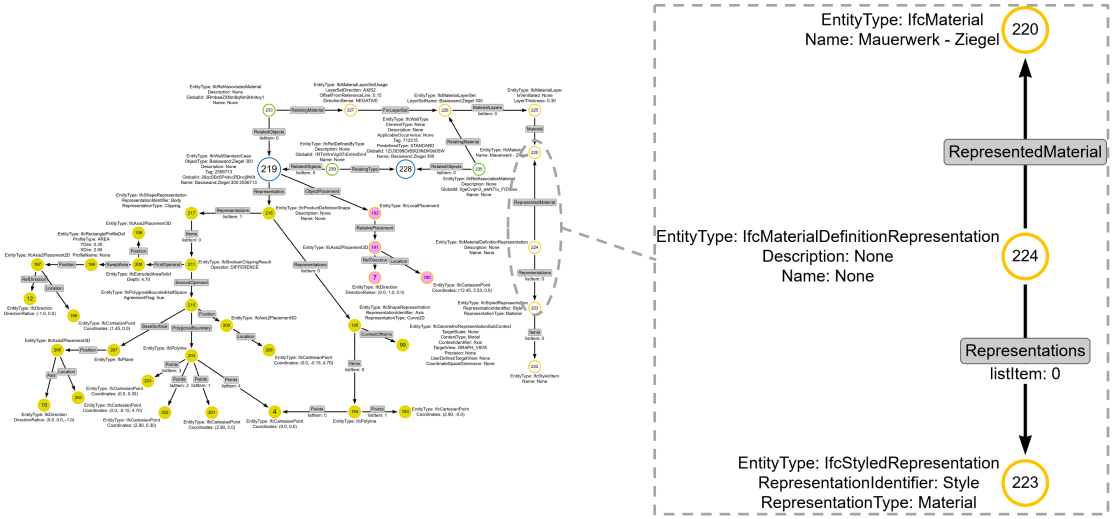


Abbildung 7.6: Auszug des PushOut-Musters des in Abschnitt 6.2.3 erläuterten Inkrements

Erkennbar ist, dass der Sekundärknoten 224 lediglich ausgehende Kanten besitzt. Demnach kann der Knoten mit der gewählten Tiefentraversierung nie erreicht werden. Die Rekursion würde in diesem Fall bei Knoten 220 stoppen, da dieser Knoten über keine ausgehenden Kanten verfügt. Die Abwesenheit eingehender Kanten auf Knoten 224 lässt sich auch mit der Spezifikation der Entität im IFC-Datenmodell begründen. Betrachtet man die EXPRESS-Definition für diese Entität, fällt auf, dass keine inversen Attribute definiert sind. Damit können keine eingehenden Kanten auf Instanzen beziehungsweise Knoten dieser Entitäten zeigen. Auf Basis der Vererbungsstruktur innerhalb des IFC-Datenmodells werden Instanzen der Entität *IfcMaterialDefinitionRepresentation* als Sekundärknoten

[1] https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/link/ifcmaterialdefinitionrepresentation.htm (Letzter Zugriff: 14.03.2024)

modelliert. Da damit weder ein eindeutiges, persistentes Attribut für den Abgleich zweier Instanzen dieser Entität vorhanden ist und die topologische Struktur die Erreichbarkeit während der Traversierung verhindert, gibt es mit dem gewählten Ansatz keine Möglichkeit, diese Knoten und mögliche anschließende Subgraphen korrekt zu traverieren und Äquivalenzen zu detektieren. Strukturell weisen diese Knoten zudem topologische Ähnlichkeiten zur Funktionsweise von Beziehungsknoten auf.

Da diese Knoten aufgrund der Unerreichbarkeit keine Äquivalenzkante erhalten können und damit niemals Teil des gemeinsamen Subgraphs G_{MCS} werden, würden diese Knoten bei der Ableitung des Inkrements als topologische Änderung behandelt und in jedem Inkrement erneut ausgetauscht werden. Diese Tatsache stellt sicher, dass die Knoten und möglicherweise an ihnen vorgenommenen Änderungen immer ausgetauscht werden. Nachteilig ist aber gleichwohl festzuhalten, dass die erstellten Inkremente damit mehr Information übermitteln als möglicherweise nötig wäre.

7.3 Lösungsansätze für Objektmodelle ohne persistente Identifikationsmerkmale

Alle in dieser Arbeit ausgeführten Überlegungen basieren auf der Annahme, dass Bauteile über verschiedene Versionen durch eindeutige Merkmale persistent angesprochen werden können. Diese Annahme konnte für viele Datenmodelle und Exportschnittstellen erfolgreich verwendet werden, allerdings existieren auch Datenrepräsentationen, die diese Charakteristik nur teilweise oder gar nicht umsetzen. Um diese Repräsentationen ebenfalls mit den erläuterten Verfahren zu verwalten, bedarf es weitergehender Untersuchungen, wie mit Daten umgegangen werden kann, die keine persistenten Identifikationsmerkmale aufweisen. Dafür sind Mechanismen notwendig, die unveränderte Teilgraphen durch geeignete Mustersuchen oder dem Einsatz von Transferwissen identifizieren können. Die kurz in Abschnitt 3.2.4 angeschnittenen Verfahren zu Graph-Neural-Networks stellen hierfür einen innovativen Ansatz dar. Mithilfe solcher Verfahren könnte eine vorgelagerte Aufbereitung der Datensätze ermöglicht werden, bei denen als äquivalent eingestufte Objekte erneut ein persistentes Merkmal bekommen können. Auf dieser Basis kann anschließend die dargelegte Prozesskette erneut zur Abstraktion vorgenommener Änderungen und deren Austausch mit anderen Projektbeteiligten realisiert werden.

7.4 Einordnung und Zusammenfassung

Zusammenfassend kann festgehalten werden, dass das entwickelte System die anvisierten Funktionalitäten der vollständigen Übertragung von Modelländerungen zwischen verschiedenen Parteien umfassend erfüllt und damit einen wesentlichen Beitrag zur Verbesserung modellbasierter Kollaboration leistet. In allen erläuterten Grenzfällen ist die Methodik imstande, vorgenommene Änderungen passend zu abstrahieren und in Form eines Inkrements mit anderen Akteuren auszutauschen. Dennoch sind insbesondere im

Umgang mit der Versionsüberwachung geometrischer Repräsentationen weitere Potenziale erkennbar, die in zukünftigen Initiativen untersucht werden sollten. Ebenso liefert die gewählte Strategie zur Ermittlung der unveränderten Teile zwischen zwei Versionen nicht immer einen möglichst großen gemeinsamen Subgraphen G_{MCS}. In der Analogie zum bekannten Versionskontrollsytem Git lässt sich aber festhalten, dass zahlreiche Grundfunktionen für ein Versionskontrollsystem und dem inkrementellen Austausch von Änderungen erfolgreich entwickelt und umgesetzt wurden. Aspekte zur inhaltlichen Konsistenz sind dabei nur am Rande behandelt worden, da sie keine Kernfunktionalität der entwickelten Methodik darstellen. Vielmehr sollten derartige Prüfungen durch spezialisierte Applikationen gewährleistet werden, die eine entsprechende Interpretation der überwachten Datenmenge ermöglichen. Applikationen wie Solibri verfügen bereits über essentielle Grundfunktionalitäten zur Untersuchung inhaltlicher Konsistenzfragen und sollten mit der zunehmenden Etablierung der Versionsthematik entsprechend erweitert werden.

Kapitel 8

Zusammenfassung und Ausblick

Die vorliegende Arbeit hat aufgezeigt, dass die bestehenden Prinzipien zur modellbasierten Kollaborationen insbesondere in der umfänglichen Verwaltung verschiedener Modellversionen Unzulänglichkeiten aufweisen. Wenngleich kollaborative Methoden auf Basis von Disziplinmodellen und deren Integration in Koordinationsmodelle immer mehr Anwendung finden, standen Herausforderungen in Bezug auf die Erfassung und Weitergabe von Änderungen in einzelnen Modellen bisher kaum im Fokus wissenschaftlicher Betrachtungen.

In dieser Arbeit wurde diese Lücke adressiert und mithilfe von graphbasierten Ansätzen ein flexibler Ansatz zur Repräsentation von Produktmodellen sowie der Versionsverwaltung mit inkrementellen Repräsentationen dargelegt. Modelle sollten zukünftig nicht mehr als monolithische, dateibasierte Repräsentationen aufgefasst werden, sondern Änderungen als Inkrement übertragen werden. Die derzeit im Bauwesen genutzten Datenmodelle bieten dafür eine hervorragende Grundlage und ermöglichen deren Versionsverwaltung auf der Basis einzelner Objekte und deren Beziehungen, die innerhalb eines Modells kodiert sind. Damit können vorgenommene Änderungen präzise erfasst und ausgetauscht werden. Gleichzeitig erhöht sich die Zugänglichkeit zu den tatsächlich geänderten Modellinformationen erheblich und trägt dazu bei, den Fokus in anschließenden Prozessen exakt auf die modifizierten Teile eines Modells zu lenken.

Durch die Verwendung von LPG-Graphen wurde eine allgemeine und flexible Darstellung der in BIM-Modellen gespeicherten Informationen gefunden. Die Anwendbarkeit wurde insbesondere an Disziplinmodellen diskutiert, die das etablierte IFC-Datenmodell als Grundlage verwenden. Die vorgestellte Methodik zur Ermittlung und Verwaltung von Modellinkrementen bedient sich weitreichender Konzepte der Graphtheorie sowie zugehöriger Transformationsmechanismen. Das entwickelte System verknüpft diese theoretische Grundlagen mit den Belangen, die insbesondere aufgrund des iterativen und nach wie vor heterogenen Charakters des Bauwesens mit vielen hochspezialisierten Disziplinen auftreten. Durch die Ermittlung aller vorgenommenen semantischen und topologischen Änderungen ausgehend von zwei Modellversionen wird sichergestellt, dass der Ansatz nicht von spezifischen Softwareprodukten abhängig ist, sondern flexibel für zahlreiche herstellerneutrale Datenstrukturen verwendet werden kann. Mit den entwickelten Mechanismen zur Identifikation unveränderter Bestandteile eines Modells auf Basis der zugehörigen Graphrepräsentationen werden anschließend semantische und topologische Modifikationen erfasst und in einem Inkrement gespeichert. Durch die verlustfreie Übertragung und Anwendung des Inkrements auf überholte Modelle wird sichergestellt, dass alle Beteiligten stets direkten Zugriff auf die vorgenommenen Änderungen haben und diese gegebenenfalls in weitere Prozesse und Auswertungen einbinden können. Durch die

Aufteilung der topologischen Transformationen in Kontext-, Klebe- und PushOut-Muster ist es gelungen, Änderungen präzise zu beschreiben und gleichzeitig eine Interpretation der Strukturen im Kontext von BIM-Modellen sicherzustellen. Dies unterstützt insbesondere bei der Verwaltung divergierender Modellzustände in unterschiedlichen Entwicklungszweigen sowie deren Zusammenführung in konsolidierte Modelle. Schließlich können die Inkremente zu jedem Zeitpunkt in die Double Push Out (DPO) Notation überführt werden, die ein weit verbreitetes allgemeines Konzept zur Graphtransformation darstellt.

Mit der entwickelten Methodik wurden wichtige Grundlagen dargelegt, die die modellbasierte Zusammenarbeit im Bauwesen für zukünftige, noch höherfrequente Austauschszenarien vorbereitet. Die Nutzer des Systems können selbstständig entscheiden, in welchem zeitlichen Intervall Änderungen übertragen werden und welche Informationen in einem Inkrement berücksichtigt werden sollen. Gleichzeitig können empfangende Nutzer überholte und veraltete Modellversionen flexibel mithilfe der Inkremente aktualisieren und diese bei Bedarf zurück in serialisierte Formen übersetzen. Damit ist eine direkte Integration des Systems in derzeit etablierte Systeme gewährleistet. Zudem kann das System in unterschiedlichen Ausprägungen zur zeitlichen Synchronisierung genutzt werden. Einerseits kann mit kurzen Intervallen eine Quasi-Echtzeitkommunikation erzielt werden. Andererseits sind aber auch lange Zeiträume des entkoppelten Arbeitens und der Synchronisation der Inkremente zu wenigen Meilensteinen in einem Projekt ohne Einschränkungen möglich. Hervorzuheben ist abschließend, dass die Kontrolle zur Erstellung eines Versionsschrittes sowie deren Synchronisation mit dem zentralen Kollaborations-Hub stets in der Eigenverantwortung des jeweiligen Nutzers liegt. Zusätzlich bleibt in allen geschilderten Ausprägungen die Verantwortung bei einem Planer verortet und ergänzt die bisherigen Methoden und Grundsätze, die im Kontext modellbasierter Zusammenarbeit heutzutage eingesetzt werden.

Im Kontext des konzipierten Kollaborations-Hubs wurden weitergehende Ansätze zum Umgang mit interdisziplinären Abhängigkeiten vorgestellt. Zusätzlich zu bekannten Funktionalitäten von CDE-Plattformen wie der Visualisierung von Modellen bildet der Verknüpfungsgraph eine wesentliche Komponente, mit der interdisziplinäre Abhängigkeiten flexibel und feingranular modelliert werden können. Auf Basis der hinzugefügten Verknüpfungen wird es möglich, Auswirkungen von Modelländerungen auf andere Disziplinen und Beteiligte deutlich früher zu erkennen und im Falle aufkommender Konflikte zügig notwendige Maßnahmen einleiten zu können. Die Vision, neben dem Transfer der Inkremente zusätzliche Überwachungs- und Benachrichtigungsmechanismen sowie fortgeschrittene Automatisierungen vorzusehen, können dabei die nächsten Schritte hin zu einer noch vernetzteren und gleichzeitig weiterhin selbstbestimmten Arbeitsweise in der interdisziplinären Welt des Bauwesens sein. Eine noch nicht abschließend gelöste Herausforderung bildet in diesem Kontext allerdings die Ermittlung der interdisziplinären Verknüpfungen von Objekten verschiedener Disziplinen. Zukünftig können hierzu möglicherweise Methoden des künstlichen Intelligenz unterstützen und bisherige Ansätze BIM-basierter Anfragesprachen um heuristische Prinzipien erweitern.

Darüber hinaus bleibt abzuwarten, in welche Richtung sich die heutzutage genutzten Datenmodelle weiterentwickeln werden. Die Erweiterungsinitiativen, die unter dem Dach von BuildingSMART International (bSI) und Open Geospatial Consortium (OGC) aktuell forciert werden, zeigen deutlich, dass noch nicht alle Bedarfe abgedeckt sind, die Akteure an die Nutzung von Modellen stellen. Insbesondere die Industry Foundation Classes wurden im letzten Jahrzehnt um eine beträchtliche Zahl an Konzepten, Entitäten und Attributen erweitert, deren vollständige und korrekte Interpretation eine zunehmend komplexe Aufgabe in der Softwareentwicklung darstellt. Ähnliche Entwicklungen sind auch im Bereich der Stadtmodelle rund um die GML-Derivate zu beobachten. Wenngleich alle Erweiterungen neue Nutzungsszenarien und genauere Beschreibungen von Sachverhalten ermöglichen, mehren sich zunehmend Stimmen, die sich für eine Entschlackung und Modularisierung der genannten Datenmodelle aussprechen. Ebenso bleibt abzuwarten, inwiefern zukünftig neue Formen der Informationskodierung zum Einsatz kommen und die derzeit weit verbreiteten Methoden wie XML, JSON oder auch STEP ablösen werden. Unstrittig ist gewiss, dass die bedarfsgerechte Zugänglichkeit von Informationen in modularer Art und Weise immer größere Bedeutung gewinnen wird, um auch komplexe Modelle und Sachverhalte schneller, einfacher und zielorientierter auf vielfältigen Endgeräten zugänglich zu machen.

Obwohl noch weitere Forschung erforderlich ist und kontinuierliche Verbesserungen an den Systemen vorgenommen werden müssen, bleibt am Ende die Zuversicht, dass diese Arbeit einen wertvollen Beitrag zur fortlaufenden Weiterentwicklung digitaler Methoden im Bauwesen leistet und somit die Akzeptanz und Anwendung dieser Technologien weiter vorantreibt.

Literaturverzeichnis

Abuoda, G. ; Dell'Aglio, D. ; Keen, A. ; & Hose, K. (2022). Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches. *QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs*, *3279*, 17–32. https://ceur-ws.org/Vol-3279/paper2.pdf

Alankarage, S., Chileshe, N., Samaraweera, A., Rameezdeen, R., & Edwards, D. J. (2022). Organisational BIM maturity models and their applications: a systematic literature review. *Architectural Engineering and Design Management*, 1–19. https://doi.org/10.1080/17452007.2022.2068496

Amann, J. (2018). *Eine objektorientierte Sprache zur Einbettung von Interpretationssemantik in digitale Bauwerksmodelle* [Diss., Technische Universität München].

Amann, J., Esser, S., Krijnen, T., Abualdenien, J., Preidel, C., & Borrmann, A. (2021). BIM-Programmierschnittstellen. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 263–290). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_13

Amann, J., Jubierre, J. R., Borrmann, A., & Flurl, M. (2014). An alignment meta-model for the comparison of alignment product models. *Proceedings of the 10th European Conference on Product and Process Modelling*, 351–358.

Attrill, R., & Mickovski, S. B. (2020). Issues to be addressed with current BIM adoption prior to the implementation of BIM level 3. *Proceedings of the 36th Annual ARCOM Conference*, 336–345. https://edshare.gcu.ac.uk/id/eprint/5179

Austern, G., Bloch, T., & Abulafia, Y. (2024). Incorporating Context into BIM-Derived Data—Leveraging Graph Neural Networks for Building Element Classification. *Buildings*, *14*(2), 527. https://doi.org/10.3390/buildings14020527

Awwad, K. A., Shibani, A., & Ghostin, M. (2022). Exploring the critical success factors influencing BIM level 2 implementation in the UK construction industry: the case of SMEs. *International Journal of Construction Management*, *22*(10), 1894–1901. https://doi.org/10.1080/15623599.2020.1744213

Bauderon, M., & Jacquet, H. (1997). Node rewriting in hypergraphs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (S. 31–43, Bd. 1197 LNCS). https://doi.org/10.1007/3-540-62559-3_4

Beck, S. F. (2023). *Context-sensitive linking of heterogeneous information models from the building and the urban domain* [Diss., Technical University of Munich].

Beetz, J., van Leeuwen, J., & de Vries, B. (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *23*(1), 89–101. https://doi.org/10.1017/S0890060409000122

Berners-Lee, T., & Connolly, D. (2004). Delta: an ontology for the distribution of differences between RDF graphs. *World Wide Web*, *4*.

Berners-Lee, T., Fielding, R. T., & Masinter, L. M. (2005). RFC 3986: Uniform Resource Identifier (URI): Generic Syntax [Letzter Zugriff: 09.03.2024]. https://datatracker.ietf.org/doc/html/rfc3986

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, *284*(5), 34–43. https://doi.org/10.1038/scientificamerican0501-34

Bew, M., & Richards, M. (2008). BIM maturity model. *Construct IT Autumn 2008 Members' Meeting. Brighton, UK*.

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

Bollobás, B. (1998). *Modern Graph Theory* (Bd. 184). Springer New York. https://doi.org/10.1007/978-1-4612-0619-4

Bonduel, M. (2021, Mai). *A Framework for a Linked Data-based Heritage BIM* [Diss., KU Leuven].

Borrmann, A., König, M., Koch, C., & Beetz, J. (2021a). *Building Information Modeling* (A. Borrmann, M. König, C. Koch & J. Beetz, Hrsg.). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4

Borrmann, A., König, M., Koch, C., & Beetz, J. (2021b). Die BIM-Methode im Überblick. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling* (2. Auflage, S. 1–31). Springer Fachmedien. https://doi.org/10.1007/978-3-658-33361-4_1

Borrmann, A., Muhic, S., Hyvärinen, J., Chipman, T., Jaud, S., Castaing, C., Dumoulin, C., Liebich, T., & Mol, L. (2019). The IFC-Bridge project – Extending the IFC standard to enable high-quality exchange of bridge information models. *2019 European Conference on Computing in Construction*, 377–386. https://doi.org/10.35490/EC3.2019.193

Borrmann, A., Rives, M., Muhic, S., Wikström, L., & Weil, J. (2022). The IFC-Tunnel project – Extending the IFC standard to enable high-quality exchange of tunnel information models. *Proc. of the International Conference on Computing in Civil and Building Engineering (ICCCBE)*.

Borrmann, A., Schorr, M., Obergriesser, M., Ji, Y., Wu, I.-C., Günthner, W., Euringer, T., & Rank, E. (2009). Using Product Data Management Systems for Civil Engineering Projects — Potentials and Obstacles. *Computing in Civil Engineering (2009)*, *346*, 359–369. https://doi.org/10.1061/41052(346)36

Braun, A. (2020). *Automated BIM-based construction progress monitoring by processing and matching semantic and geometric data* [Diss., Technische Universität München].

Breunig, M., Borrmann, A., Rank, E., Hinz, S., Kolbe, T., Schilcher, M., Mundani, R.-P., Jubierre, J. R., Flurl, M., Thomsen, A., Donaubauer, A., Ji, Y., Urban, S., Laun, S., Vilgertshofer, S., Willenborg, B., Menninghaus, M., Steuer, H., Wursthorn, S., … Mazroobsemnani, N. (2017). Collaborative Multi-Scale 3D City and Infrastruc-

ture Modeling and Simulation. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-4/W4*, 341–352. https://doi.org/10.5194/isprs-archives-XLII-4-W4-341-2017

British Standards Institution. (2013). *PAS 1192-2:2013 : specification for information management for the capital/delivery phase of construction projects using building information modelling*.

Bryan, P., & Nottingham, M. (2013). RFC 6902: JavaScript Object Notation (JSON) Patch [Letzter Zugriff: 09.01.2024]. https://tools.ietf.org/html/rfc6902

Bryan, P., Zyp, K., & Nottingham, M. (2013). RFC 6901: JavaScript Object Notation (JSON) Pointer [Letzter Zugriff: 09.01.2024]. https://tools.ietf.org/html/rfc6902

Bucher, D. F., & Hall, D. M. (2020). Common Data Environment within the AEC Ecosystem: moving collaborative platforms beyond the open versus losed dichotomy. *EG-ICE 2020 Proceedings: Workshop on Intelligent Computing in Engineering*, 491–500. https://doi.org/10.3929/ethz-b-000447240

Buder, J. (2017). *Neues Planungsverfahren für Anlagen der Leit- und Sicherungstechnik auf Basis durchgängiger elektronischer Datenhaltung* [Diss., Technische Universität Dresden].

Bunke, H. (1997). On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, *18*(8), 689–694. https://doi.org/10.1016/S0167-8655(97)00060-3

Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., & Vento, M. (2002). A comparison of algorithms for maximum common subgraph on randomly connected graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2396*, 123–132. https://doi.org/10.1007/3-540-70659-3_12

Busse, J., Humm, B., Lübbert, C., Moelter, F., Reibold, A., Rewald, M., Schlüter, V., Seiler, B., Tegtmeier, E., & Zeh, T. (2014). Was bedeutet eigentlich Ontologie? *Informatik-Spektrum*, *37*(4), 286–297. https://doi.org/10.1007/s00287-012-0619-2

CEN. (2018). DIN EN ISO 19650-1:2018 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — Information management using building information modelling — Part 1: Concepts and principles. https://doi.org/10.31030/3030494

Chacon, S. (2009). *Pro Git*. Apress. https://doi.org/10.1007/978-1-4302-1834-0

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., & Widom, J. (1996). Change Detection in Hierarchically Structured Information. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, *25*(2), 493–504. https://doi.org/10.1145/235968.233366

Collins, F. C., Ringsquandl, M., Braun, A., Hall, D. M., & Borrmann, A. (2022). Shape encoding for semantic healing of design models and knowledge transfer to scan-to-BIM. *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, *175*(4), 160–180. https://doi.org/10.1680/jsmic.21.00032

Conte, D., Foggia, P., & Vento, M. (2007). Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a

wide database of graphs. *Journal of Graph Algorithms and Applications*, *11*(1), 99–143. https://doi.org/10.7155/jgaa.00139

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd Edition). The MIT Press.

Crockford, D. (2006). RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON) [Letzter Zugriff: 09.03.2024]. https://datatracker.ietf.org/doc/html/rfc4627

Daum, S. (2018). *Konzeption einer raum-zeitlichen Anfragesprache für die Analyse und Prüfung von 4D-Gebäudeinformationsmodellen* [Diss., Technische Universität München].

Dayal, U., Hsut, M., & Ladins, R. (1990). Organizing Long-Running Activities with Triggers and Transactions. *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, *45*(3), 204–214.

Deutsches Institut für Normung e. V. (2019a). 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. https://doi.org/10.31030/3044839

Deutsches Institut für Normung e.V. (2018). DIN 1356-1: Bauzeichnungen –Teil 1: Grundregeln der Darstellung. https://doi.org/10.31030/2430882

Deutsches Institut für Normung e.V. (2019b). DIN SPEC 91391-1: Gemeinsame Datenumgebungen (CDE) für BIM-Projekte-Funktionen und offener Datenaustausch zwischen Plattformen unterschiedlicher Hersteller-Teil 1: Module und Funktionen einer Gemeinsamen Datenumgebung. https://doi.org/10.31030/3044838

Diestel, R. (2017). *Graph Theory* (5. Auflage). Springer Spektrum Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-53622-3

Digital Built Britain. (2015). Level 3 Building Information Modelling - Strategic Plan [Letzter Aufruf: 09.03.2024]. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/410096/bis-15-155-digital-built-britain-level-3-strategy.pdf

Donkers, A., Yang, D., & Baken, N. (2020). Linked Data for Smart Homes: Comparing RDF and Labeled Property Graphs. *LDAC2020—8th Linked Data in Architecture and Construction Workshop*, 23–36.

Donkers, A., Yang, D., & de Vries, B. (2023). Semantic web-enabled outlier and missing value detection and replacement in smart buildings. *2023 European Conference on Computing in Construction*, 172. https://doi.org/10.35490/EC3.2023.172

Dörr, H. (1995). Graph rewriting systems — The basic concepts. In H. Dörr (Hrsg.), *Efficient Graph Rewriting and Its Implementation. Lecture Notes in Computer Science* (922. Aufl., S. 9–33). Springer. https://doi.org/10.1007/BFb0031911

Dürst, M. J., & Suignard, M. (2005). RFC 3987: Internationalized Resource Identifiers (IRIs) [Letzter Zugriff: 09.03.2024]. https://datatracker.ietf.org/doc/html/rfc3987

Eastman, C. (1999). Information Exchange Architectures for Building Models (M. Lacasse & D. Vanier, Hrsg.). *Durability of Building Materials and Components*, *8*, 2139–2156.

Eastman, C., Fisher, D., Lafue, G., Lividini, J., Stoker, D., & Yessios, C. (1974). An Outline of the Building Description System [Research Report no. 50].

Eastman, C., & Augenbroe, F. (1998). Product Modeling Strategies for Today and the Future. *Proc. of the CIB W78 conference The life-cycle of construction IT innovations*, 191–208.

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors* (Bd. 2. Auflage). John Wiley & Sons.

Ebertshäuser, S., Brüggemann, T., & von Both, P. (2021). 3D-Stadtmodellierung: CityGML. In *Building Information Modeling* (S. 243–261, Bd. 2. Auflage). Springer. https://doi.org/10.1007/978-3-658-33361-4_12

ECMA International. (2017). The JSON Data Interchange Syntax [letzter Zugriff: 09.03.2024]. https://ecma-international.org/publications-and-standards/standards/ecma-404/

Eichhoff, J. R., & Roller, D. (2016). Designing the Same, but in Different Ways: Determinism in Graph-Rewriting Systems for Function-Based Design Synthesis. *Journal of Computing and Information Science in Engineering*, *16*(1). https://doi.org/10.1115/1.4032576

Eilebrecht, K., & Starke, G. (2019). *Patterns kompakt*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-57937-4

Fernandes, D., & Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. *Proceedings of the 7th International Conference on Data Science, Technology and Applications*, 373–380. https://doi.org/10.5220/0006910203730380

Firmenich, B., Koch, C., Richter, T., & Beer, D. G. (2005). Versioning structured object sets using text based Version Control Systems. *Proceedings of the 22nd CIB-W78 conference*.

Fischer, M., & Froese, T. (1992). Integration through Standard Project Models. In R. T. Vanier D (Hrsg.), *Computers and information in construction*. http://itc.scix.net/paper/w78-1992-189

Fischer, S., Schranz, C., Urban, H., & Pfeiffer, D. (2023). Automation of escape route analysis for BIM-based building code checking. *Automation in Construction*, *156*, 105092. https://doi.org/10.1016/j.autcon.2023.105092

Flurl, M., Mundani, R.-P., & Rank, E. (2014). Graph-based concurrency control for multi-scale procedural models. *Proceedings of the 10th European Conference on Product and Process Modelling, ECPPM 2014*, 319–325.

Forth, K., Abualdenien, J., & Borrmann, A. (2023). Calculation of embodied GHG emissions in early building design stages using BIM and NLP-based semantic model healing. *Energy and Buildings*, *284*, 112837. https://doi.org/10.1016/j.enbuild.2023.112837

Foulds, L. R. (1992). *Graph Theory Applications*. Springer New York. https://doi.org/10.1007/978-1-4612-0933-1

Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A., & Gauthier, P. (1997). Cluster-based scalable network services. *Proceedings of the 16th ACM symposium on Operating systems principles*, 78–91. https://doi.org/10.1145/268998.266662

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., & Taylor, A. (2018). Cypher: An Evolving Query Language for Property Graphs. *Proceedings of the 2018 International Conference on Management of Data*, 1433–1445. https://doi.org/10.1145/3183713.3190657

Fraser, N. (2009). Differential synchronization. *DocEng'09 - Proceedings of the 2009 ACM Symposium on Document Engineering*, 13–20. https://doi.org/10.1145/1600193.1600198

Gabler Wirtschaftslexikon. (2022). Datenmodell Definition [Letzter Zugriff: 09.03.2024]. https://wirtschaftslexikon.gabler.de/definition/datenmodell-28093/version-251730

Gallagher, B. (2006). Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. *AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection (Vol. 45)*, 43–53.

Geiß, R. (2007). *Graphersetzung mit Anwendungen im Übersetzerbau* [Diss., Universität Fridericiana zu Karlsruhe].

Gelling, E., Fletcher, G., & Schmidt, M. (2023). Bridging graph data models: RDF, RDF-star, and property graphs as directed acyclic graphs. http://arxiv.org/abs/2304.13097

Göbels, A. (2022). Enabling object-based documentation of existing bridge inspection data using Linked Data. *Proceedings of the 33. Forum Bauinformatik*, 148–155.

Goldfarb, C. F. (1981). A generalized approach to document markup. *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, 68–73. https://doi.org/10.1145/800209.806456

Grudin, J. (1994). Computer-Supported Cooperative Work: History and Focus. *Computer*, *27*(5), 19–26. https://doi.org/10.1109/2.291294

Guyo, E. D., Hartmann, T., & Snyders, S. (2023). An ontology to represent firefighters data requirements during building fire emergencies. *Advanced Engineering Informatics*, *56*, 101992. https://doi.org/10.1016/j.aei.2023.101992

Habel, A., Müller, J., & Plump, D. (2001). *Double-pushout graph transformation revisited* (Bd. 11). https://doi.org/10.1017/S0960129501003425

Haerder, T., & Reuter, A. (1983). Principles of Transaction-Oriented Database Recovery. *ACM computing surveys (CSUR)*, *15*(4), 287–317.

Han, L., Ge, G., & M., G. (2023). A Parallel IFC Normalization Algorithm for Incremental Storage and Version Control. *Proc. of the 30th Int. Conference on Intelligent Computing in Engineering (EG-ICE)*. https://www.ucl.ac.uk/bartlett/construction/sites/bartlett_construction/files/a_parallel_ifc_normalization_algorithm_for_incremental_storage_and_version_control.pdf

Haralambous, Y. (2007). *Fonts & encodings* (First Edition). O'Reilly Media.

Harold, E. R., & Means, W. S. (2004). *XML in a Nutshell* (3rd Edition). O'Reilly Media.

Hars, A. (1994). *Referenzdatenmodelle*. Gabler Verlag. https://doi.org/10.1007/978-3-322-90397-6

Hartmann, T., Amor, R., & East, E. W. (2017). Information Model Purposes in Building and Facility Design. *Journal of Computing in Civil Engineering*, *31*(6), 1–10. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000706

Heckel, R. (2006). Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science*, *148*(1), 187–198. https://doi.org/10.1016/j.entcs.2005.12.018

Heckel, R., & Taentzer, G. (2020). *Graph Transformation for Software Engineers*. Springer International Publishing. https://doi.org/10.1007/978-3-030-43916-3

Helms, B., & Shea, K. (2012). Computational Synthesis of Product Architectures Based on Object-Oriented Graph Grammars. *Journal of Mechanical Design*, *134*(2). https://doi.org/10.1115/1.4005592

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

Hunt, J. W., & Mcilroy, M. D. (1976). An Algorithm for Differential File Comparison. *Computing Science Technical Report*, *41*.

Ismail, A., Nahar, A., & Scherer, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard. *24th International Workshop on Intelligent Computing in Engineering (EG-ICE 2017)*.

ISO. (1994a). ISO 10303-1:1994: Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles. https://www.iso.org/standard/72237.html

ISO. (1994b). ISO 10303-11:1994: Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual. https://www.iso.org/standard/38047.html

ISO. (1998). ISO/IEC 8859-1: Information technology - 8-bit single-byte coded graphic character sets. Part 1: Latin alphabet No. 1.

ISO. (2016). ISO 10303-21:2016: Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure (ISO 10303-21:1994). https://www.iso.org/standard/78580.html

ISO. (2019a). ISO 10303-44:2019: Industrial automation systems and integration — Product data representation and exchange — Part 44: Integrated generic resource: Product structure configuration. https://www.iso.org/standard/78580.html

ISO. (2019b). ISO 19162:2019: Geographic Information Well-Known Text Representation of Coordinate Reference Systems. https://www.iso.org/standard/76496.html

ISO. (2020). ISO 19136-1:2020: Geographic information — Geography Markup Language (GML) — Part 1: Fundamentals.

Jakumeit, E., Buchwald, S., & Kroll, M. (2010). GrGen.NET. *International Journal on Software Tools for Technology Transfer*, *12*(3-4), 263–271. https://doi.org/10.1007/s10009-010-0148-8

Jaskula, K., Papadonikolaki, E., & Rovas, D. (2023). Comparison of current common data environment tools in the construction industry. *2023 European Conference*

on Computing in Construction & 40th International CIB W78 Conference. https://doi.org/10.35490/EC3.2023.315

Jaud, Š., Esser, S., Muhič, S., & Borrmann, A. (2020). Development of IFC schema for infrastructure. *Proceedings of the 6th International Conference siBIM: Structured Data are New Gold*, 27–35.

Johansson, J., Contero, M., Company, P., & Elgh, F. (2018). Supporting connectivism in knowledge based engineering with graph theory, filtering techniques and model quality assurance. *Advanced Engineering Informatics*, *38*, 252–263. https://doi.org/10.1016/j.aei.2018.07.005

Katz, R. H. (1990). Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, *22*(4), 375–409. https://doi.org/10.1145/98163.98172

Kay, A. C. (1996). The early history of Smalltalk. In *History of programming languages—II* (S. 511–598).

Kielar, P. M., Biedermann, D. H., Kneidl, A., & Borrmann, A. (2018). A unified pedestrian routing model for graph-based wayfinding built on cognitive principles. *Transportmetrica A: Transport Science*, *14*(5), 406–432. https://doi.org/10.1080/23249935.2017.1309472

Kiviniemi, A., Fischer, M., & Bazjanac, V. (2005). Integration of Multiple Product Models: IFC Model Servers as a Potential Solution. *Proc. of the 22nd CIB-W78 Conference on Information Technology in Construction*, 37–40.

Kneidl, A., Borrmann, A., & Hartmann, D. (2012). Generation and use of sparse navigation graphs for microscopic pedestrian simulation models. *Advanced Engineering Informatics*, *26*(4), 669–680. https://doi.org/10.1016/j.aei.2012.03.006

Kniemeyer, O. G. M. (2008). Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. *PhD Thesis*, (November 2008), 432. papers2://publication/uuid/6B64424B-60E4-4042-BC27-C53E0B188AA0

Kobler, M. (2010). *Qualität von Prozessmodellen: Kennzahlen zur analytischen Qualitätssicherung bei der Prozessmodellierung*. Logos Verlag Berlin GmbH.

Koch, C., & Firmenich, B. (2011). An approach to distributed building modeling on the basis of versions and changes. *Advanced Engineering Informatics*, *25*(2), 297–310. https://doi.org/10.1016/j.aei.2010.12.001

Kolbeck, L., Vilgertshofer, S., Abualdenien, J., & Borrmann, A. (2022). Graph Rewriting Techniques in Engineering Design. *Front. Built Environment*, *7*, 1–19. https://doi.org/10.3389/fbuil.2021.815153

König, M., Rahm, T., Nagel, F., & Ludger, S. (2017). BIM-Anwendungen im Tunnelbau. *Ernst & Sohn Verlag für Architektur und technische Wissenschaften GmbH & Co. KG Bautechnik 94 (2017), Heft 4*, 227–231. https://doi.org/10.1002/bate.201700005

Kurul, E., Oti, A. H., & Cheung, F. K. T. (2016). Level 3 BIM for Standardised Design Delivery, Refinement and Optimisation: Is it a real option in the UK? *CIB World Building Congress*.

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, *88*(1), 43–61. https://doi.org/10.1007/s41064-020-00095-z

Kwon, S., Monnier, L. V., Barbau, R., & Bernstein, W. Z. (2020). Enriching standards-based digital thread by fusing as-designed and as-inspected data using knowledge graphs. *Advanced Engineering Informatics*, *46*, 101102. https://doi.org/10.1016/j.aei.2020.101102

Lee, G., Won, J., Ham, S., & Shin, Y. (2011). Metrics for Quantifying the Similarities and Differences between IFC Files. *Journal of Computing in Civil Engineering*, *25*(2), 172–181. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000077

Lin, X., Chen, J., Lu, W., & Guo, H. (2024). An edge-weighted graph triumvirate to represent modular building layouts. *Automation in Construction*, *157*, 105140. https://doi.org/10.1016/j.autcon.2023.105140

Lindholm, T. (2004). A three-way merge for XML documents. *Proceedings of the 2004 ACM symposium on Document engineering*, 1–10. https://doi.org/10.1145/1030397.1030399

Luo, L., He, Q., Jaselskis, E. J., & Xie, J. (2017). Construction Project Complexity: Research Trends and Implications. *Journal of Construction Engineering and Management*, *143*(7). https://doi.org/10.1061/(ASCE)CO.1943-7862.0001306

McCreesh, C., Prosser, P., & Trimble, J. (2017). A Partitioning Algorithm for Maximum Common Subgraph Problems. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 712–719. https://doi.org/10.24963/ijcai.2017/99

Meinhardt, P., Knuth, M., & Sack, H. (2015). TailR: A Platform for Preserving History on the Web of Data. *Proceedings of the 11th International Conference on Semantic Systems*, 57–64. https://doi.org/10.1145/2814864.2814875

Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, *28*(5), 449–462. https://doi.org/10.1109/TSE.2002.1000449

Myers, E. W. (1986). An O(ND) difference algorithm and its variations. *Algorithmica*, *1*, 251–266. https://doi.org/10.1007/BF01840446

Nash, A., Huerlimann, D., Schuette, J., & Krauss, V. P. (2010). RailML - A standard data interface for railroad applications. *Timetable Planning and Information Quality*, 3–10. https://doi.org/10.2495/978-1-84564-500-7/01

Nguyen, S. H., & Kolbe, T. H. (2020). A MULTI-PERSPECTIVE APPROACH TO INTERPRETING SPATIO-SEMANTIC CHANGES OF LARGE 3D CITY MODELS IN CITYGML USING A GRAPH DATABASE. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *VI-4/W1-2020*, 143–150. https://doi.org/10.5194/isprs-annals-VI-4-W1-2020-143-2020

Nguyen, S. H., & Kolbe, T. H. (2022). PATH-TRACING SEMANTIC NETWORKS TO INTERPRET CHANGES IN SEMANTIC 3D CITY MODELS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *X-4/W2-2022*, 217–224. https://doi.org/10.5194/isprs-annals-X-4-W2-2022-217-2022

Nour, M. (2009). Performance of different (BIM/IFC) exchange formats within a private collaborative workspace for collaborative work. *Journal of Information Technology in Construction*, *14*, 736–752. http://www.itcon.org/2009/48

Nour, M., & Beucke, K. (2008). An Open Platform for Processing IFC Model Versions. *Tsinghua Science and Technology*, *13*(SUPPL. 1), 126–131. https://doi.org/10.1016/S1007-0214(08)70138-X

Nour, M., Firmenich, B., Richter, T., & Koch, C. (2006). A Versioned IFC Datbase for Multi-Disciplinary Synchronous Cooperation. *Joint International Conference on Computing and Decision Making in Civil and Building Engineering*, 636–645. https://itc.scix.net/pdfs/w78-2006-tf111.pdf

Nuha, M. U. (2019). *Data Versioning for Graph Databases* [Masterarbeit, Delft University of Technology].

Object Management Group. (2017). OMG Unified Modeling Language (OMG UML). https://www.omg.org/spec/UML/2.5.1/PDF

Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D., & Christophides, V. (2013). High-level change detection in RDF(S) KBs. *ACM Transactions on Database Systems*, *38*(1), 1–42. https://doi.org/10.1145/2445583.2445584

Pauwels, P., Costin, A., & Rasmussen, M. H. (2022). Knowledge Graphs and Linked Data for the Built Environment. In M. Bolpagni, R. Gavina & D. Ribeiro (Hrsg.), *Industry 4.0 for the Built Environment: Methodologies, Technologies and Skills* (S. 157–183). Springer International Publishing. https://doi.org/10.1007/978-3-030-82430-3_7

Pauwels, P., & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, *63*, 100–133. https://doi.org/10.1016/j.autcon.2015.12.003

Pauwels, P., Terkaj, W., Krijnen, T., & Beetz, J. (2015). Coping with lists in the ifcOWL ontology. *Proceedings of the 22nd International Workshop on Intelligent Computing in Engineering*.

Pfaltz, J. L., & Rosenfeld, A. (1970). Web Grammars. *Proceedings of the First International Joint Conference on Artificial Intelligence*, 609–619.

Poinet, P., Stefanescu, D., Tsakiridis, G., De Boissieu, A., & Papadonikolaki, E. (2020). Supporting collaborative design and project management for AEC using Speckle's interactive data flow diagram. *DC I/O 2020*. https://doi.org/10.47330/DCIO.2020.VYBA4375

Preidel, C. (2020). *Automatisierte Konformitätsprüfung digitaler Bauwerksmodelle hinsichtlich geltender Normen und Richtlinien mit Hilfe einer visuellen Programmiersprache* [Diss., Technische Universität München].

Preidel, C., Borrmann, A., Exner, H., & König, M. (2021). Common Data Environment. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling* (2. Auflage, S. 335–351). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_16

Rasmussen, M. H., Lefrançois, M., Bonduel, M., Hviid, C. A., & Karlshøj, J. (2018). OPM: An ontology for describing properties that evolve over time. *Proceedings of the 6th Linked Data in Architecture and Construction Workshop*, 24–33.

Rasmussen, M. H., Lefrançois, M., Pauwels, P., Hviid, C. A., & Karlshøj, J. (2019). Managing interrelated project information in AEC Knowledge Graphs. *Automation in Construction*, *108*, 102956. https://doi.org/10.1016/j.autcon.2019.102956

Rasmussen, M. H., Lefrançois, M., Schneider, G. F., & Pauwels, P. (2020). BOT: The building topology ontology of the W3C linked building data group. *Semantic Web*, *12*, 143–161. https://doi.org/10.3233/SW-200385

Ray, E. T., & McIntosh, J. (2002). *Perl and XML*. O'Reilly Media.

Rebolj, D., Tibaut, A., Čuš-Babič, N., Magdič, A., & Podbreznik, P. (2008). Development and application of a road product model. *Automation in Construction*, *17*(6), 719–728. https://doi.org/10.1016/j.autcon.2007.12.004

Richter, T. (2009). *Konzepte für den Einsatz versionierter Objektmodelle im Bauwesen* [Diss., Bauhaus-Universität Weimar].

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases* (M. Beaugureau, Hrsg.). O'Reilly Media. https://doi.org/10.1016/b978-0-12-407192-6.00003-0

Rochkind, M. J. (1975). The source code control system. *IEEE Transactions on Software Engineering*, *SE-1*(4), 364–370. https://doi.org/10.1109/TSE.1975.6312866

Sakr, S., Wylot, M., Mutharaju, R., Le Phuoc, D., & Fundulaki, I. (2018). *Linked Data*. Springer International Publishing. https://doi.org/10.1007/978-3-319-73515-3

Schapke, S.-E., Beetz, J., König, M., Koch, C., & Borrmann, A. (2021). Prinzipien und Techniken der modellgestützten Zusammenarbeit. In A. Borrmann, M. König, C. Koch & J. Beetz (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 309–333). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_15

Schilling, A. (2018). *Architektur und Modellbau: Konzepte, Methoden, Materialien*. Birkhäuser. https://doi.org/10.1515/9783035614749

Schlenger, J., Yeung, T., Vilgertshofer, S., Martinez, J., Sacks, R., & Borrmann, A. (2022). A Comprehensive Data Schema for Digital Twin Construction. *Proceedings of the 29th EG-ICE International Workshop on Intelligent Computing in Engineering*, 34–44. https://doi.org/10.7146/aul.455.c194

Sciore, E. (1994). Versioning and Configuration Management in an Object-Oriented Data Model. *The VLDB journal*, *3*, 77–106.

Semenov, V., & Jones, S. (2015). IFChub: Multimodal Collaboration Transactional Guarantees Platform with Solid Transactional Guarantees. *Proc. of the 32nd CIB W78 Conference 2015, 27th-29th October 2015, Eindhoven, The Netherlands*, 667–675.

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

Slepicka, M., Vilgertshofer, S., & Borrmann, A. (2022). Fabrication information modeling: interfacing building information modeling with digital fabrication. *Construction Robotics*. https://doi.org/10.1007/s41693-022-00075-2

Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Springer-Verlag. https://archive.org/details/Stachowiak1973AllgemeineModelltheorie

Succar, B. (2010). Building Information Modelling Maturity Matrix. In J. Underwood & U. Isikdag (Hrsg.), *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies* (S. 65–103). IGI Global. https://doi.org/10.4018/978-1-60566-928-1.ch004

Sutherland, I. E. (1963). SketchPad: A man-machine graphical communication system. *AFIPS Conference Proceedings 23*, 323–328.

Tauscher, E., Bargstädt, H.-J., & Smarsly, K. (2016). Generic BIM queries based on the IFC object model using graph theory. *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, 6–14.

Teclaw, W., Rasmussen, M. H., Labonnote, N., Oraskari, J., & Hjelseth, E. (2023). The semantic link between domain-based BIM models. *Proceedings of the 11th Linked Data in Architecture and Construction Workshop*. https://hdl.handle.net/11250/3101194

The Unicode Consortium. (2022). The Unicode Standard [Letzter Zugriff: 09.03.2024]. https://www.unicode.org/versions/Unicode15.1.0/

Tichy, W. F. (1984). *RCS: A System for Version Control* (Techn. Ber. Nr. 84-474). Department of Computer Science Technical Reports. https://docs.lib.purdue.edu/cstech/394

Turk, Ž. (1999). Constraints of Product Modelling Approach in Building. *8th International Conference on Durability of Building*.

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

Urpalainen, J. (2008). RFC 5261: An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors [Letzter Zugriff: 09.03.2024]. https://datatracker.ietf.org/doc/html/rfc5261

Uschold, M., & Gruninger, M. (1996). Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, *11*(2), 93–136. https://doi.org/10.1017/S0269888900007797

Vajna, S., Weber, C., Zeman, K., Hehenberger, P., Gerhard, D., & Wartzack, S. (2018). CAx-Systeme – warum und wozu? In *CAx für Ingenieure* (S. 1–23). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54624-6_1

van Berlo, L., Krijnen, T., Tauscher, H., van Kranenburg, A., & Paasiala, P. (2021). Future of the Industry Foundation Classes: towards IFC 5. *Proceedings of the 38th International Conference of CIB W78*, 123–137.

van Nederveen, G., & Tolman, F. (1992). Modelling multiple views on buildings. *Automation in Construction*, *1*(3), 215–224. https://doi.org/10.1016/0926-5805(92)90014-B

Verein Deutscher Ingenieure. (2020). VDI 2552 Blatt 1: Building Information Modeling Grundlagen [Letzter Zugriff: 09.03.2024]. https://www.vdi.de/richtlinien/details/vdi-2552-blatt-1-building-information-modeling-grundlagen

Vilgertshofer, S. (2022). *Kopplung von Graphersetzung und parametrischer Modellierung zur Unterstützung des modellbasierten Entwerfens und der Erstellung mehrskaliger Modelle* [Diss., Technische Universität München].

Vilgertshofer, S., & Borrmann, A. (2017). Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models. *Advanced Engineering Informatics*, *33*, 502–515. https://doi.org/10.1016/j.aei.2017.07.003

Voss, C., Petzold, F., & Rudolph, S. (2023). Graph transformation in engineering design: an overview of the last decade. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *37*, e5. https://doi.org/10.1017/S089006042200018X

W3C. (2014). RDF 1.1 Concepts and Abstract Syntax.

Wagner, A., Sprenger, W., Maurer, C., Kuhn, T. E., & Rüppel, U. (2022). Building product ontology: Core ontology for Linked Building Product Data. *Automation in Construction*, *133*, 103927. https://doi.org/10.1016/j.autcon.2021.103927

Walmsley, P. (2001). *Definitive XML schema*. Pearson Education.

Wang, Y., & Maple, C. (2005). A novel efficient algorithm for determining maximum common subgraphs. *Proceedings of the International Conference on Information Visualisation*, *2005*, 657–663. https://doi.org/10.1109/IV.2005.11

Weidinger, J. (2022). *Deep Learning based integration of manual changes on floor plans* [Masterarbeit, Technische Universität München].

Weise, M. (2006). *Ein Ansatz zur Abbildung von Änderungen in der modell-basierten Objektplanung* [Diss., Technische Universitaet Dresden].

Weise, M., Katranuschkov, P., & Scherer, R. (2003). Generalised model subset definition schema. *Proceedings of the 20th CIB-W78 Conference on Information Technology in Construction*.

Weise, M., Katranuschkov, P., & Scherer, R. (2004). Managing long transactions in model server based collaboration. *Proceedings of the 5th European Conference on Product and Process Modelling in the AEC industry*, 311.

Willenbacher, H. (2002). *Interaktive verknüpfungsbasierte Bauwerksmodellierung* [Diss., Bauhaus-Universität Weimar].

Wunsch, S., & Jaekel, B. (2017). Modellprinzipien des RailTopoModel. *Der Eisenbahningenieur*, *3*, 18–23.

Zeller, A., & Snelting, G. (1997). Unified versioning through feature logic. *ACM Transactions on Software Engineering and Methodology*, *6*(4), 398–441. https://doi.org/10.1145/261640.261654

Zhu, J., Wu, P., & Lei, X. (2023). IFC-graph for facilitating building information access and query. *Automation in Construction*, *148*, 104778. https://doi.org/10.1016/j.autcon.2023.104778

Anhang A

Bezeichnung des Anhangs

A.1 Darstellung der zum Anwendungsbeispiel gehörenden Cypher-Befehle

A.1.1 Inkrement zur Modifikation der Stützen im Tragwerksmodell

Algorithmus A.1 beschreibt die semantischen Modifikationen, die im Inkrement $\delta_{12,TW}$ enthalten sind und in Abschnitt 6.2.1 erläutert wurden. In der ersten Variante müssen die einzelnen Paare aus *MATCH* und *SET* nacheinander in individuellen Anfragen ausgeführt werden, um keine Konflikte in der Benennung der Knoten hervorzurufen.

Algorithmus A.1: Semantische Modifikationen der im Commit $\delta_{12,TW}$ erfassten Modelländerungen

```
MATCH
(A:ts20240215T144400:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"3cEe0uSgr1VBLkaQJil8gU"})
SET A.Description = "300*300"
// ---
MATCH
(A:ts20240215T144400:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"3cEe0uSgr1VBLkaQJil8gU"})
SET A.Name = "STB Stütze-rechteckig:STB300 x 300:2520640"
// ---
MATCH
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"3cEe0uSgr1VBLkaQJil8gU"})
-[:rel{rel_type:"Representation"}]->
(B:SecondaryNode:ts20240215T144400{EntityType:"IfcProductDefinitionShape"})
-[:rel{listItem:0, rel_type:"Representations"}]->
(C:SecondaryNode:ts20240215T144400{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(D:SecondaryNode:ts20240215T144400{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(E:SecondaryNode:ts20240215T144400{EntityType:"IfcArbitraryClosedProfileDef
    "})
SET E.ProfileName = "300*300"
// ---
MATCH
(A:ts20240215T144400:PrimaryNode{
    EntityType:"IfcColumn",
```

```
    GlobalId:"3cEe0uSgr1VBLkaQJil8gU"})
-[:rel{rel_type:"Representation"}]->
(B:SecondaryNode:ts20240215T144400{EntityType:"IfcProductDefinitionShape"})
-[:rel{listItem:0, rel_type:"Representations"}]->
(C:SecondaryNode:ts20240215T144400{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(D:SecondaryNode:ts20240215T144400{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(E:SecondaryNode:ts20240215T144400{EntityType:"IfcArbitraryClosedProfileDef
    "})
-[:rel{rel_type:"OuterCurve"}]->
(F:SecondaryNode:ts20240215T144400{EntityType:"IfcIndexedPolyCurve"})
-[:rel{rel_type:"Points"}]->
(G:SecondaryNode:ts20240215T144400{EntityType:"IfcCartesianPointList2D"})
SET G.CoordList = "((-200.0, -200.0), (200.0, -200.0), (200.0, 200.0),
    (-200.0, 200.0))"
// ---
MATCH
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"1OjOzH5vf9th7m6ACPB3dG"})
SET A.Description = "400*400"
// ---
MATCH
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"1OjOzH5vf9th7m6ACPB3dG"})
SET A.Name = "STB Stütze - rechteckig:STB 400 x 400:2521453"
// ---
MATCH
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"3iENKWgi9A2PmPzh89eSv2"})
SET A.Description = "400*400"
// ---
MATCH
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"3iENKWgi9A2PmPzh89eSv2"})
SET A.Name = "STB Stütze - rechteckig:STB 400 x 400:2521492"
```

Die genannten Befehle können in beliebiger Reihenfolge ausgeführt werden und könnten bei Bedarf weitergehend zusammengefasst werden. Als verkürzte Variante könnte je Stütze auch immer ein Muster definiert werden, dass alle zu modifizierenden Knoten beinhaltet, und anschließend alle *SET*-Befehle ausführt. Daraus ergäben sich die in Algorithmus A.2, Algorithmus A.3 und Algorithmus A.4 gezeigten verkürzten Varianten, die in Anhang A.1.1 dargestellt werden.

Algorithmus A.2: Aggregierte Form der im Commit $\delta_{12,TW}$ erfassten Modelländerungen für Stütze mit GlobalId 3cEe0uSgr1VBLkaQJil8gU

```
MATCH path0 =
(A:ts20240215T144400:PrimaryNode{
```

```
    EntityType:"IfcColumn",
    GlobalId:"3cEe0uSgr1VBLkaQJil8gU"})
-[:rel{rel_type:"Representation"}]->
(B:SecondaryNode:ts20240215T144400{EntityType:"IfcProductDefinitionShape"})
-[:rel{listItem:0, rel_type:"Representations"}]->
(C:SecondaryNode:ts20240215T144400{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(D:SecondaryNode:ts20240215T144400{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(E:SecondaryNode:ts20240215T144400{EntityType:"IfcArbitraryClosedProfileDef
    "})
-[:rel{rel_type:"OuterCurve"}]->
(F:SecondaryNode:ts20240215T144400{EntityType:"IfcIndexedPolyCurve"})
-[:rel{rel_type:"Points"}]->
(G:SecondaryNode:ts20240215T144400{EntityType:"IfcCartesianPointList2D"})

SET A.Description = "300*300"
SET A.Name = "STB Stütze-rechteckig:STB300 x 300:2520640"
SET E.ProfileName = "300*300"
SET G.CoordList = "((-200.0, -200.0), (200.0, -200.0), (200.0, 200.0),
    (-200.0, 200.0))"
```

Algorithmus A.3: Aggregierte Form der im Commit $\delta_{12,TW}$ erfassten Modelländerungen für Stütze mit GlobalId 1OjOzH5vf9th7m6ACPB3dG

```
MATCH path0 =
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"1OjOzH5vf9th7m6ACPB3dG"})
SET A.Description = "400*400"
SET A.Name = "STB Stütze - rechteckig:STB 400 x 400:2521453"
```

Algorithmus A.4: Aggregierte Form der im Commit $\delta_{12,TW}$ erfassten Modelländerungen für Stütze mit GlobalId 3iENKWgi9A2PmPzh89eSv2

```
MATCH path0 =
(A:PrimaryNode:ts20240215T144400{
    EntityType:"IfcColumn",
    GlobalId:"3iENKWgi9A2PmPzh89eSv2"})
SET A.Description = "400*400"
SET A.Name = "STB Stütze - rechteckig:STB 400 x 400:2521492"
```

A.1.2 Inkrement zur Modifikation der Stützen im Architekturmodell

Algorithmus A.5 zeigt das Cypher-Statement zur Anwendung der im Inkrement $\delta_{12,ARC}$ enthaltenen Modifikationen, die in Abschnitt 6.2.2 erläutert wurden. Zu Zwecken der Übersichtlichkeit wurden die einzelnen *MATCH*- und *SET*-Befehle gruppiert und aggregiert.

Algorithmus A.5: Anwendung der in $\delta_{12,ARC}$ enthaltenen Modelländerungen in einer aggregierten Darstellung

```
MATCH (n189:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"0h0LzMJt93cez1WHNgxod6"})
-[:rel{rel_type:"Representation"}]->
(n186:ts20240220T112536:SecondaryNode{EntityType:"IfcProductDefinitionShape
    "})
-[:rel{listItem:0, rel_type:"Representations"}]->
(n185:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n184:ts20240220T112536:SecondaryNode{EntityType:"IfcMappedItem"})
-[:rel{rel_type:"MappingSource"}]->
(n182:ts20240220T112536:SecondaryNode{EntityType:"IfcRepresentationMap"})
-[:rel{rel_type:"MappedRepresentation"}]->
(n180:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n178:ts20240220T112536:SecondaryNode{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(n175:ts20240220T112536:SecondaryNode{EntityType:"IfcRectangleProfileDef"})
-[:rel{rel_type:"Position"}]->
(n174:ts20240220T112536:SecondaryNode{EntityType:"IfcAxis2Placement2D"})
-[:rel{rel_type:"Location"}]->
(n173:ts20240220T112536:SecondaryNode{EntityType:"IfcCartesianPoint"})

MATCH (n170:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"0h0LzMJt93cez1WHNgxoa"})
-[:rel{rel_type:"Representation"}]->
(n167:ts20240220T112536:SecondaryNode{EntityType:"IfcProductDefinitionShape
    "})
-[:rel{listItem:0, rel_type:"Representations"}]->
(n166:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n165:ts20240220T112536:SecondaryNode{EntityType:"IfcMappedItem"})
-[:rel{rel_type:"MappingSource"}]->
(n163:ts20240220T112536:SecondaryNode{EntityType:"IfcRepresentationMap"})
-[:rel{rel_type:"MappedRepresentation"}]->
(n161:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n159:ts20240220T112536:SecondaryNode{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(n156:ts20240220T112536:SecondaryNode{EntityType:"IfcRectangleProfileDef"})
-[:rel{rel_type:"Position"}]->
(n155:ts20240220T112536:SecondaryNode{EntityType:"IfcAxis2Placement2D"})
-[:rel{rel_type:"Location"}]->
(n154:ts20240220T112536:SecondaryNode{EntityType:"IfcCartesianPoint"})

MATCH (n151:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"2A$LC1Lgn0qvSYKm11Vune"})
-[:rel{rel_type:"Representation"}]->
(n148:ts20240220T112536:SecondaryNode{EntityType:"IfcProductDefinitionShape
    "})
-[:rel{listItem:0, rel_type:"Representations"}]->
```

```
(n147:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n146:ts20240220T112536:SecondaryNode{EntityType:"IfcMappedItem"})
-[:rel{rel_type:"MappingSource"}]->
(n138:ts20240220T112536:SecondaryNode{EntityType:"IfcRepresentationMap"})
-[:rel{rel_type:"MappedRepresentation"}]->
(n136:ts20240220T112536:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[:rel{listItem:0, rel_type:"Items"}]->
(n130:ts20240220T112536:SecondaryNode{EntityType:"IfcExtrudedAreaSolid"})
-[:rel{rel_type:"SweptArea"}]->
(n127:ts20240220T112536:SecondaryNode{EntityType:"IfcRectangleProfileDef"})
-[:rel{rel_type:"Position"}]->
(n126:ts20240220T112536:SecondaryNode{EntityType:"IfcAxis2Placement2D"})
-[:rel{rel_type:"Location"}]->
(n125:ts20240220T112536:SecondaryNode{EntityType:"IfcCartesianPoint"})

MATCH (n183:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumnType",
    GlobalId:"0h0LzMJt93cez1YHNgxod6"})
MATCH (n164:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumnType",
    GlobalId:"0h0LzMJt93cez1YHNgxoa$"})
MATCH (n139:ts20240220T112536:PrimaryNode{
    EntityType:"IfcColumnType",
    GlobalId:"2A$LC1Lgn0qvSYMm11Vune"})

SET n189.ObjectType = "STB Stütze - rechteckig:STB 400 x 400"
SET n189.Name = "STB Stütze - rechteckig:STB 400 x 400:2521492"
SET n175.YDim = 0.4
SET n175.XDim = 0.4
SET n175.ProfileName = "STB 400 x 400"
SET n173.Coordinates = "(0.0, 2.1649348980190553e-15)"

SET n170.ObjectType = "STB Stütze - rechteckig:STB 400 x 400"
SET n170.Name = "STB Stütze - rechteckig:STB 400 x 400:2521453"
SET n156.YDim = 0.4
SET n156.XDim = 0.4
SET n156.ProfileName = "STB 400 x 400"
SET n154.Coordinates = "(0.0, 0.0)"

SET n151.ObjectType = "STB Stütze - rechteckig:STB 400 x 400"
SET n151.Name = "STB Stütze - rechteckig:STB 400 x 400:2520640"
SET n127.YDim = 0.4
SET n127.XDim = 0.4
SET n127.ProfileName = "STB 400 x 400"
SET n125.Coordinates = "(0.0, 0.0)"

SET n183.Tag = "2077224"
SET n183.Name = "STB Stütze - rechteckig:STB 400 x 400"

SET n164.Tag = "2077224"
SET n164.Name = "STB Stütze - rechteckig:STB 400 x 400"
```

```
SET n139.Tag = "2077224"
SET n139.Name = "STB Stütze - rechteckig:STB 400 x 400"
```

A.1.3 Inkrement zum Hinzufügen der Innenwand im Architekturmodell

Algorithmus A.6 zeigt den zugehörigen Code, um die zum Einfügen der Wand gehörende topologische Modifikation des Graphs durchzuführen. Die zugehörigen Erklärungen finden sich in Abschnitt 6.2.3.

Algorithmus A.6: Anwendung der in $\delta_{23,ARC}$ enthaltenen Modelländerungen

```
// --- --- --- --- --- --- Kontext-Muster --- --- --- --- --- ---

MATCH path0 =
(n103:ts20240220T112845:PrimaryNode{
    EntityType:"IfcProject",
    GlobalId:"1ODmFv4Jv9ZO9fO_v2Tu_8"})
-[e888:rel{rel_type:"OwnerHistory"}]->
(n18:SecondaryNode:ts20240220T112845{EntityType:"IfcOwnerHistory"})

MATCH path1 =
(n189:ts20240220T112845:PrimaryNode{
    EntityType:"IfcColumn",
    GlobalId:"0h0LzMJt93cez1WHNgxod6"})
-[e765:rel{rel_type:"Representation"}]->
(n186:SecondaryNode:ts20240220T112845{EntityType:"IfcProductDefinitionShape
    "})
-[e769:rel{listItem:0, rel_type:"Representations"}]->
(n185:ts20240220T112845:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[e770:rel{rel_type:"ContextOfItems"}]->
(n100:SecondaryNode:ts20240220T112845{EntityType:"
    IfcGeometricRepresentationSubContext"})

MATCH path2 =
(n185)
-[e771:rel{listItem:0, rel_type:"Items"}]->
(n184:SecondaryNode:ts20240220T112845{EntityType:"IfcMappedItem"})
-[e772:rel{rel_type:"MappingSource"}]->
(n182:ts20240220T112845:SecondaryNode{EntityType:"IfcRepresentationMap"})
-[e777:rel{rel_type:"MappedRepresentation"}]->
(n180:ts20240220T112845:SecondaryNode{EntityType:"IfcShapeRepresentation"})
-[e780:rel{listItem:0, rel_type:"Items"}]->
(n178:SecondaryNode:ts20240220T112845{EntityType:"IfcExtrudedAreaSolid"})
-[e784:rel{rel_type:"Position"}]->
(n177:SecondaryNode:ts20240220T112845{EntityType:"IfcAxis2Placement3D"})
-[e788:rel{rel_type:"RefDirection"}]->
(n6:ts20240220T112845:SecondaryNode{EntityType:"IfcDirection"})

MATCH path3 =
(n178)
-[e785:rel{rel_type:"ExtrudedDirection"}]->
(n9:ts20240220T112845:SecondaryNode{EntityType:"IfcDirection"})
```

```
MATCH path4 =
(n103)
-[e890:rel{listItem:0, rel_type:"RepresentationContexts"}]->
(n98:SecondaryNode:ts20240220T112845{EntityType:"
    IfcGeometricRepresentationContext"})
-[e895:rel{rel_type:"WorldCoordinateSystem"}]->
(n96:SecondaryNode:ts20240220T112845{EntityType:"IfcAxis2Placement3D"})
-[e897:rel{rel_type:"Location"}]->
(n3:SecondaryNode:ts20240220T112845{EntityType:"IfcCartesianPoint"})

MATCH path5 =
(n229:ts20240220T112601:ConnectionNode{
    EntityType:"IfcRelContainedInSpatialStructure", GlobalId:"13
        LV5dTeP3CAox54$56C1Z"})

// --- --- --- --- --- --- Push-Out-Muster --- --- --- --- --- ---

MERGE
(n219:ts20240220T112845:PrimaryNode{
    EntityType:"IfcWallStandardCase",
    ObjectType:"Basiswand:Ziegel 300",
    Description:"None",
    Tag:"2556713",
    GlobalId:"26qcSBz5P4dvcRDncj9N0t",
    Name:"Basiswand:Ziegel 300:2556713"})
-[e727:rel{relType:"Representation"}]->
(n218:SecondaryNode:ts20240220T112845{
    EntityType:"IfcProductDefinitionShape",
    Description:"None",
    Name:"None"})

MERGE (n218)
-[e729:rel{listItem:1, relType:"Representations"}]->
(n217:ts20240220T112845:SecondaryNode{
    EntityType:"IfcShapeRepresentation",
    RepresentationIdentifier:"Body",
    RepresentationType:"Clipping"})

MERGE (n217)
-[e731:rel{listItem:0, relType:"Items"}]->
(n211:SecondaryNode:ts20240220T112845{
    EntityType:"IfcBooleanClippingResult",
    Operator:"DIFFERENCE"})

MERGE (n211)
-[e738:rel{relType:"SecondOperand"}]->
(n210:SecondaryNode:ts20240220T112845{
    EntityType:"IfcPolygonalBoundedHalfSpace",
    AgreementFlag:True})

MERGE (n210)
-[e741:rel{relType:"PolygonalBoundary"}]->
```

```
(n204:SecondaryNode:ts20240220T112845{
    EntityType:"IfcPolyline"})

MERGE (n204)
-[e750:rel{listItem:3, relType:"Points"}]->
(n203:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(0.0, 0.30)"})

MERGE (n204)
-[e749:rel{listItem:2, relType:"Points"}]->
(n202:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(2.90, 0.30)"})

MERGE (n204)
-[e748:rel{listItem:1, relType:"Points"}]->
(n201:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(2.90, 0.0)"})

MERGE (n204)
-[e747:rel{listItem:4, relType:"Points"}]->
(n4:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(0.0, 0.0)"})

MERGE (n210)
-[e740:rel{relType:"Position"}]->
(n209:SecondaryNode:ts20240220T112845{
    EntityType:"IfcAxis2Placement3D"})

MERGE (n209)
-[e742:rel{relType:"Location"}]->
(n208:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(0.0, -0.15, 4.70)"})

MERGE (n210)
-[e739:rel{relType:"BaseSurface"}]->
(n207:SecondaryNode:ts20240220T112845{
    EntityType:"IfcPlane"})

MERGE (n207)
-[e743:rel{relType:"Position"}]->
(n206:SecondaryNode:ts20240220T112845{
    EntityType:"IfcAxis2Placement3D"})

MERGE (n206)
-[e745:rel{relType:"Axis"}]->
(n10:ts20240220T112845:SecondaryNode{
    EntityType:"IfcDirection",
    DirectionRatios:"(0.0, 0.0, -1.0)"})
```

```
MERGE (n206)
-[e744:rel{relType:"Location"}]->
(n205:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(0.0, -0.15, 4.70)"})

MERGE (n211)
-[e737:rel{relType:"FirstOperand"}]->
(n200:SecondaryNode:ts20240220T112845{
    EntityType:"IfcExtrudedAreaSolid",
    Depth:4.70})

MERGE (n200)
-[e752:rel{relType:"Position"}]->
(n199:SecondaryNode:ts20240220T112845{
    EntityType:"IfcAxis2Placement3D"})

MERGE (n200)
-[e751:rel{relType:"SweptArea"}]->
(n198:SecondaryNode:ts20240220T112845{
    EntityType:"IfcRectangleProfileDef",
    ProfileType:"AREA",
    YDim:0.30,
    XDim:2.90,
    ProfileName:"None"})

MERGE (n198)
-[e755:rel{relType:"Position"}]->
(n197:SecondaryNode:ts20240220T112845{
    EntityType:"IfcAxis2Placement2D"})

MERGE (n197)
-[e757:rel{relType:"RefDirection"}]->
(n12:ts20240220T112845:SecondaryNode{
    EntityType:"IfcDirection",
    DirectionRatios:"(-1.0, 0.0)"})

MERGE (n197)
-[e756:rel{relType:"Location"}]->
    (n196:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(1.45, 0.0)"})

MERGE (n218)
-[e728:rel{listItem:0, relType:"Representations"}]->
(n195:ts20240220T112845:SecondaryNode{
    EntityType:"IfcShapeRepresentation",
    RepresentationIdentifier:"Axis",
    RepresentationType:"Curve2D"})

MERGE (n195)
-[e1056:rel{listItem:0, relType:"Items"}]->
```

```
(n194:SecondaryNode:ts20240220T112845{
    EntityType:"IfcPolyline"})

MERGE (n194)
-[e1059:rel{listItem:1, relType:"Points"}]->
(n193:SecondaryNode:ts20240220T112845{
    EntityType:"IfcCartesianPoint",
    Coordinates:"(2.90, -0.0)"})

MERGE (n194)
-[e1058:rel{listItem:0, relType:"Points"}]->
(n4)

MERGE (n195)
-[e1055:rel{relType:"ContextOfItems"}]->
(n99:SecondaryNode:ts20240220T112845{
    EntityType:"IfcGeometricRepresentationSubContext",
    TargetScale:"None",
    ContextType:"Model",
    ContextIdentifier:"Axis",
    TargetView:"GRAPH_VIEW",
    Precision:"None"
    UserDefinedTargetView:"None",
    CoordinateSpaceDimension:"None"})

MERGE (n219)
-[e726:rel{relType:"ObjectPlacement"}]->
(n192:SecondaryNode:ts20240220T112845{
    EntityType:"IfcLocalPlacement"})

MERGE (n192)
-[e759:rel{relType:"RelativePlacement"}]->
(n191:SecondaryNode:ts20240220T112845{
    EntityType:"IfcAxis2Placement3D"})

MERGE (n191)
-[e762:rel{relType:"RefDirection"}]->
(n7:ts20240220T112845:SecondaryNode{
    EntityType:"IfcDirection",
    DirectionRatios:"(0.0, 1.0, 0.0)"})

MERGE (n191)
-[e760:rel{relType:"Location"}]->
(n190:SecondaryNode:ts20240220T112845{
EntityType:"IfcCartesianPoint",
    Coordinates:"(12.45, 5.54, 0.0)"})

MERGE (n233:ConnectionNode:ts20240220T112845{
    EntityType:"IfcRelAssociatesMaterial",
    Description:"None",
    GlobalId:"3RmbaeZXNm8qNm9HnItoy1",
    Name:"None"})
-[e697:rel{listItem:0, relType:"RelatedObjects"}]->
```

```
(n219)

MERGE (n233)
-[e696:rel{relType:"RelatingMaterial"}]->
(n227:SecondaryNode:ts20240220T112845{
    EntityType:"IfcMaterialLayerSetUsage",
    LayerSetDirection:"AXIS2",
    OffsetFromReferenceLine:0.15,
    DirectionSense:"NEGATIVE"})

MERGE (n239:ConnectionNode:ts20240220T112845{
    EntityType:"IfcRelDefinesByType",
    Description:"None",
    GlobalId:"1RTnKtmVgl37rEnHo$VnfZ",
    Name:"None"})
-[e674:rel{listItem:0, relType:"RelatedObjects"}]->
(n219)

MERGE (n239)
-[e673:rel{relType:"RelatingType"}]->
(n228:ts20240220T112845:PrimaryNode{
    EntityType:"IfcWallType",
    ElementType:"None",
    Description:"None",
    ApplicableOccurrence:"None",
    Tag:"712215",
    PredefinedType:"STANDARD",
    GlobalId:"1ZU3D9$OrB9Q3fkDK9b0SW",
    Name:"Basiswand:Ziegel 300"})

MERGE (n235:ConnectionNode:ts20240220T112845{
    EntityType:"IfcRelAssociatesMaterial",
    Description:"None",
    GlobalId:"0geCvqm3_awNTIu_FrDGao",
    Name:"None"})
-[e686:rel{listItem:0, relType:"RelatedObjects"}]->
(n228)

MERGE  (n235)
-[e685:rel{relType:"RelatingMaterial"}]->
(n226:SecondaryNode:ts20240220T112845{
    EntityType:"IfcMaterialLayerSet",
    LayerSetName:"Basiswand:Ziegel 300"})

MERGE (n227)
-[e716:rel{relType:"ForLayerSet"}]->
(n226)

MERGE (n226)
-[e717:rel{listItem:0, relType:"MaterialLayers"}]->
(n225:SecondaryNode:ts20240220T112845{
    EntityType:"IfcMaterialLayer",
    IsVentilated:"None",
```

```
    LayerThickness:0.30})

MERGE (n225)
-[e718:rel{relType:"Material"}]->
(n220:SecondaryNode:ts20240220T112845{
    EntityType:"IfcMaterial",
    Name:"Mauerwerk - Ziegel"})

MERGE (n224:SecondaryNode:ts20240220T112845{
    EntityType:"IfcMaterialDefinitionRepresentation",
    Description:"None",
    Name:"None"})
-[e719:rel{relType:"RepresentedMaterial"}]->
(n220)

MERGE (n224)
-[e720:rel{listItem:0, relType:"Representations"}]->
(n223:SecondaryNode:ts20240220T112845{
    EntityType:"IfcStyledRepresentation",
    RepresentationIdentifier:"Style",
    RepresentationType:"Material"})

MERGE (n223)
-[e722:rel{listItem:0, relType:"Items"}]->
(n222:SecondaryNode:ts20240220T112845{
    EntityType:"IfcStyledItem",
    Name:"None"})

// --- --- --- --- --- ---  Klebe-Muster --- --- --- --- --- ---
MERGE (n219)-[e725:rel{rel_type:"OwnerHistory"}]->(n18)
MERGE (n233)-[e695:rel{rel_type:"OwnerHistory"}]->(n18)
MERGE (n239)-[e672:rel{rel_type:"OwnerHistory"}]->(n18)
MERGE (n228)-[e715:rel{rel_type:"OwnerHistory"}]->(n18)
MERGE (n235)-[e684:rel{rel_type:"OwnerHistory"}]->(n18)

MERGE (n229)-[e21:rel{rel_type:"RelatedElements", listItem:23}]->(n219)

MERGE (n192)-[e758:rel{rel_type:"PlacementRelTo"}]->(n110)
MERGE (n217)-[e730:rel{rel_type:"ContextOfItems"}]->(n100)
MERGE (n206)-[e746:rel{rel_type:"RefDirection"}]->(n6)
MERGE (n199)-[e754:rel{rel_type:"Location"}]->(n3)

MERGE (n200)-[e753:rel{rel_type:"ExtrudedDirection"}]->(n9)
MERGE (n191)-[e761:rel{rel_type:"Axis"}]->(n9)

MERGE (n99)-[e894:rel{rel_type:"ParentContext"}]->(n98)
MERGE (n223)-[e721:rel{rel_type:"ContextOfItems"}]->(n98)
```